

CHAPTER 1

INTRODUCTION TO ESA 86/88-3

ESA 86/88-3 is an advanced, general-purpose microcomputer system that can be operated with 8086 or 8088 CPU and provides the user a powerful and complete environment for 8086 based programming and applications development.

8086 and 8088 are third generation CPUs from INTEL that differ primarily in their external data paths. 8088 uses an 8-bit wide data bus while 8086 uses a 16-bit wide data bus. ESA 86/88-3 can be operated with either CPU and the only possible difference would be in the speed of execution (with 8088 CPU, a small speed degradation occurs because of the 8-bit wide data bus). In either case, the CPU is operated in maximum mode.

ESA 86/88-3 can be used not only as an instructional and learning aid, but also as a development tool in R&D labs and industries. It is a versatile system and can be configured in a variety of ways, as per user requirements. The basic system can be easily expanded through system BUS connectors. Powerful features like monitor resident Symbolic One-line assembler and Disassembler simplifies the programmer's task of entering Assembly language programs. On-board provision for 8087 Numeric Data Processor makes ESA 86/88-3 useful for number-crunching applications also. Onboard battery backup provision is an added feature that takes care of RAM data in the event of inadvertent power failures. It also features an onboard DAC, 12 bit ADC (optional) and parallel printer interface.

ESA 86/88-3 can be operated in standalone mode using optional PC/AT keyboard and LCD or in serial mode with the host computer through its RS 232C/RS 485 interface.

1.1 SYSTEM CAPABILITIES

- ♦ Assemble 8086/8088 Instruction Mnemonics using ESA 86/88-3 Symbolic One-Line Assembler.
- ♦ Disassemble HEX bytes from memory into 8086/88 CPU instructions.
- ♦ Perform fast numerical computations using the optional 8087 Numeric Data Processor.
- ♦ Execute the user program at full speed or debug the program through Single Step and Breakpoint facilities.
- ♦ Examine/Modify the contents of memory locations in byte or word format.
- ♦ Examine/Modify the processor registers.
- ♦ Write or read data to or from I/O ports (byte or word format).
- ♦ Operations on blocks of memory such as filling a block of memory with a constant byte or word data, comparing a block of memory with another block, copying a block of data or program within the memory and displaying memory blocks in byte or word format.



- ♦ Communicate with a Host PC serially through RS 232C/RS485 interface at a baud rate of up to 38400 and develop/debug applications using the user-friendly Windows or DOS driver packages.
- ♦ Download user programs into ESA 86/88-3 from a host computer system in Intel HEX as well as Intel Extended HEX format.
- ♦ Upload user programs to Host Computer system and save them as HEX files on a system.
- ♦ Use the monitor resident Centronics compatible Parallel printer driver software and obtain hard copies of Serial mode operations.
- ♦ Convert 8-bit digital data to analog voltage using onboard D/A converter.
- ♦ Convert analog signals to 12-bit digital data from up to 8 independent channels using onboard A/D converter.
- ♦ Program popular EPROMs 2716 through 27512 using the EPROM Programmer software embedded in the monitor.

1.2 SPECIFICATIONS

➤ **Central Processor**

8086 or 8088 CPU - operating at 8 MHz in maximum mode (Default Supply: 8086-2 CPU).

➤ **Co-Processor**

On-board 8087 Numeric Data processor (optional)

➤ **Memory**

The memory of ESA 86/88-3 can be configured in various ways and allows the user to realize the full 1M Byte addressing capability of the CPU.

EPROM: 2 JEDEC compatible slots offer the following options:

- ♦ 128K bytes using 27512
- ♦ 256K bytes using 27010
- ♦ 512K bytes using 27020
- ♦ 1024K bytes using 27040

RAM: 2 JEDEC Compatible slots offer the following:

- ♦ 256K bytes using 628128
- ♦ 512K bytes using 628512

EXPANSION MEMORY: 2 JEDEC slots allow expansion using either ROM or RAM.



➤ **Peripherals, Controllers & Interfacing Options**

SCN 2681: Dual Universal Asynchronous Receiver/Transmitter supporting standard baud rates up to 38400, selected through on-board DIP switch setting.

8253-5: Programmable Interval Timer provides 3 independent timing channels to the user.

8255A: (5 Nos.) Programmable Peripheral Interface devices provide 120 Programmable I/O lines.

8259A: Programmable Interrupt Controller provides interrupt vectors for 8 independent sources.

8288: Bus Controller used for generating control signals.

8042 UPI: Universal Peripheral Interface for PC Keyboard Interfacing

➤ **Interrupts**

EXTERNAL

- ♦ NMI: 8086/8088 Type 2 Interrupt connected to BREAK key on the trainer. The vectoring information for this interrupt is fully user-defined.
- ♦ INTR: This line is controlled through an onboard 8259A (Programmable Interrupt Controller), which provides interrupt vectors for eight independent onboard or off-board sources.

INTERNAL: Interrupt Vectors 1 (Single step Interrupt) and 3 (Breakpoint Interrupt) reserved for monitor.

➤ **Interfacing options**

- ♦ Centronics compatible Parallel Printer interface
- ♦ Onboard 8 Channel, 12-bit A/D converter
- ♦ Onboard 8-bit D/A Converter.
- ♦ ESA EPROM Programmer Interface
- ♦ ESA Study card modules for 8255, 8279, 8259, 8251 and 8253

➤ **Interface Signals**

CPU Bus : Demultiplexed and fully buffered, TTL compatible, Address, Data & Control signals are available on two 50 pin ribbon cable connectors.

Parallel I/O : 48 programmable parallel I/O lines (TTL Compatible) through two 26-pin ribbon cable connectors that are compatible with those of other ESA trainers

Serial I/O : RS 232C through on-board 9 pin D-type female connector.
RS 485 through on board 9 pin D type male connector.

Timer Signals : Timer 0,1,2 signals are brought to a header.

Analog Signals : 8 analog inputs for ADC are fed through terminal blocks.
DAC output is available at a test point.

Power Supplies: + 5V @1.5A (max)
(Optional) ± 12V@250mA (max) for ADC, DAC.



1.3 SCOPE OF SUPPLY (Default)

- ♦ ESA 86/88-3 Trainer Kit
- ♦ ESA 86/88-3 User's Manual
- ♦ ESA 86/88 Driver Software CD
- ♦ RS 232C Cable
- ♦ MCS-86 Assembly Reference Card



CHAPTER 2

CONFIGURATION AND INSTALLATION

2.1 CONFIGURATION OF ESA 86/88-3

ESA 86/88-3 is a versatile Microprocessor trainer and can be configured in a number of ways, as determined by the Dip Switch Settings and Jumpers. (Refer to the component layout diagram in Appendix C to locate the DIP Switch and the jumpers). This chapter describes all the configuration options and the installation procedures of ESA 86/88-3 trainer.

2.1.1 PROCESSOR AND CLOCK SELECTION

ESA 86/88-3 can be operated either with an 8086 or 8088 CPU. Install the desired CPU (8086 or 8088) in the socket position U33. The following DIP Switch setting selects the appropriate CPU.

DIP SW7	Processor
OFF	8086*
ON	8088

(* Default factory installation)

The CPU can be operated at a frequency of 8MHz or 5MHz, depending on jumper JP-23. Closing this jumper will select a clock frequency of 5MHz for the CPU. Factory setting leaves the jumper open for CPU operation at 8Mhz. Note that this clock option is available only if a 15 MHz crystal has been provided on the trainer.

2.1.2 OPERATIONAL MODE SELECTION

ESA 86/88-3 can be operated either in the serial mode or in stand-alone mode. In stand-alone mode the trainer will be interfaced with the LCD display and optional PC keyboard. In the serial mode, the trainer can be connected to a CRT terminal or to a host computer system through an RS 232C or RS 485 communication standard interface. The selection of the desired mode of operation is done as follows:

DIP SW5	DIP SW8	Operational mode
OFF	X	Stand-alone mode
ON	OFF	Serial mode with RS 485 interface
ON	ON	Serial mode with RS 232C interface

Separate serial connectors are provided on the trainer for interfacing with RS 232C and RS 485 (optional) ports. The user may interface the CRT terminal or host PC using the appropriate connector.

(X = Don't care)

Chapter 4 describes the Monitor commands available for working with ESA 86/88-3 in stand-alone mode and serial mode.



2.1.3 BAUD RATE SELECTION

ESA 86/88-3 uses SCN 2681 Dual Channel UART for serial communication. Channel A is used for RS 232C communication and Channel B for RS 485 communication. Both channels of the UART are configured as follows. These initializations are made by system firmware.

- ◆ Asynchronous mode
- ◆ 8-bit character length
- ◆ Stop Bit length 2.00
- ◆ No parity
- ◆ Baud rate factor of 16 x

Baud Rate selection for Serial communication using SCN 2681 Dual Channel UART is obtained by setting Dip Switches 1 to 4. Dip Switch settings for various Baud rates are described below.

DIP SW1	DIP SW2	DIP SW3	DIP SW4	Baud Rate
ON	ON	ON	ON	50
OFF	ON	ON	ON	110
ON	OFF	ON	ON	134.5
OFF	OFF	ON	ON	200
ON	ON	OFF	ON	300
OFF	ON	OFF	ON	600
ON	OFF	OFF	ON	1200
OFF	OFF	OFF	ON	1050
ON	ON	ON	OFF	2400
OFF	ON	ON	OFF	4800
ON	OFF	ON	OFF	7200
OFF	OFF	ON	OFF	9600 ¹
ON	ON	OFF	OFF	38400 ²
OFF	ON	OFF	OFF	NOT USED
ON	OFF	OFF	OFF	NOT USED
OFF	OFF	OFF	OFF	NOT USED

¹Default factory Setting

²Max Baud rate using Windows Driver package

2.1.4 PRINTER ENABLE/DISABLE

ESA 86/88-3 firmware includes the driver software for Centronics compatible Parallel Printer interface. This driver can be enabled/disabled as shown below. Section 6.10 describes installation and operation of this interface in detail.

DIP SW6	Printer Driver
OFF*	Disabled
ON	Enabled

(*Default factory Setting)



2.1.5 MEMORY SELECTION

ESA 86/88-3 has six 32-pin sockets, labeled U6 to U11 for RAM and ROM through which the user can address up to 1M Bytes of memory. Memory configuration and jumper settings for various memory ICs is described below.

Sockets U6 & U9 are populated with EPROMs 27C010 (128K x 2 = 256K) containing system firmware. Sockets U7 & U10 are populated with RAMs 628128 (128K x 2 = 256K). This RAM can be upgraded by populating the sockets with 628512 (512K RAM) and by using appropriate jumper settings. Sockets U8 & U11 are provided for memory expansion using either RAM or ROM.

Socket Nos.	Devices Compatible	Memory Type (total)
U6 & U9	1) 27512	128K EPROM
	2) 27010	256K EPROM
	3) 27020	512K EPROM
	4) 27040	1024K EPROM
U7 & U10	1) 628128	256K SRAM
	2) 628512	512K SRAM
U8 & U11	1) 27512	128K Expansion ROM
	2) 27010	256K Expansion ROM
	3) 27020	512K Expansion ROM
	4) 628128	256K Expansion RAM
	5) 628512	512K Expansion RAM

Memory Configuration

1) EPROM

<u>Device at U6 & U9</u>	<u>JP-6</u>
27C512	B-C
27C010*	B-C
27C020	A-B
27C040	A-B

2) RAM

<u>Device at U7 & U10</u>	<u>JP-20</u>	<u>JP-8</u>
628128*	A-B	A-B
628512	B-C	B-C

3) EXPANSION MEMORY

Device at U8 & U11	JP-9	JP-10	JP-11	JP-12	JP-13	JP-14
27C512	X	A-B	X	B-C	A-B	A-B
27C010	A-B	A-B	A-B	B-C	A-B	A-B
27C020	A-B	B-C	A-B	B-C	A-B	A-B
628128	B-C	A-B	A-B	A-B	B-C	B-C
628512	B-C	B-C	B-C	A-B	B-C	B-C

(X = Don't care; * Default factory setting)



2.1.6 8087 INSTALLATION

ESA 86/88-3 has on-board provision for 8087 Numeric Data Processor. To install it, just populate the socket labeled U32 with 8087. No other hardware changes are necessary.

2.1.7 INTERRUPT SELECTION

ESA 86/88-3 features an 8259A Priority Interrupt controller. The interrupt sources to 8259A can be selected either from certain on-board signals or off-board signals. The selection can be made using the information in the following table.

JUMPER	INTERRUPT SELECTED	8259A INTERRUPT
JP-15 – B-C JP-15 – A-B	PC3 of U42 (8255A) INT0*	IR0
JP-16 – B-C JP-16 – A-B	PC3 of U31 (8255A) INT1*	IR1
JP-17 – B-C JP-17 – A-B	PC0 of U31 (8255A) INT2*	IR2
JP-1 – B-C JP-1 – A-B	TxRDYA of U12 (SCN 2681) INT3*	IR3
JP-2 – B-C JP-2 – A-B	RxRDYA of U12 (SCN 2681) INT4*	IR4
JP-3 – B-C JP-3 – A-B	TIMER 1 of U5 (8253) INT5*	IR5
JP-4 – B-C JP-4 – A-B	TIMER 2 of U5 (8253) INT6*	IR6
JP-5 – B-C JP-5 – A-B	NDPINT of U32 (8087) INT7*	IR7

(INT0* to INT7* are from system connector J1. Factory installation leaves all the jumpers open)

2.2 INSTALLATION OF ESA 86/88-3

To install and operate ESA 86/88-3 trainer, the following accessories are required

a) Power Supply

+ 5V @ 1.5 A (max)

± 12V @ 250mA (max) for working with ADC, DAC

b) For Serial mode of operation: CRT or Host PC with RS 232C/RS 485 interface with the driver software and RS 232C/RS 485 compatible Serial Cables.

c) For Standalone-mode operation: 20 x 4 LCD and PC Keyboard



The following steps are common for either mode of operation with ESA 86/88-3 Trainer

1. Select the CPU (8086 or 8088) and Mode of Operation using the DIP Switch.
2. Select interrupt sources if required by configuring the jumpers
3. Select EPROM configuration if necessary (Ref. Section 2.1.5)
4. Install 8087 NDP if desired (Ref. Section 2.1.6)
5. Connect the Power Supply of required capacity to ESA 86/88-3 (Ref Section 2.2)

For Serial Mode Operation:

1. Select Baud rate, Communication Standard (RS232C or RS 485) and Printer Enable/Disable by setting the DIP Switch
2. Connect ESA 86/88-3 to Host PC / CRT through an RS 232C/RS 485 cable. Interfacing requirements for both RS 232C and RS 485 communication are described in Appendix E.
3. The terminal/computer system should be powered ON and the driver software should be running Ref. Chapter 3 for details of installation of Driver Software and Communication procedures
4. Install the LCD if required and Power ON the trainer

Now the following sign-on message should appear on the console depending on the processor selected.

ESA 86 MONITOR Vx.y or ESA 88 MONITOR Vx.y
(V x.y indicates Version x and Revision y)

The sign-on message is followed by the command prompt, "." in the next line.

Now ESA 86/88-3 is ready for operation in Serial mode. If an LCD interface is used, the following message appears on the Display.

ESA 86 MONITOR Vx.y
SER:9600 P:86

The message on the LCD gives information about the CPU being used, the mode of operation, current baud rate configuration (in case of Serial mode) and the status of the Printer driver.

If the trainer is configured to work with 8088 CPU in Serial Mode with a baud rate of 4800 and if the printer driver is enabled, the message on the LCD upon RESET will be

ESA 86 MONITOR Vx.y
SER:4800 P:88 PRN

No response in serial mode:

If there is no response from ESA 86/88-3 in Serial mode, after installation as described above, check the following.

- ♦ Power supply connections as required.
- ♦ Configuration of ESA 86/88-3 (DIP Switch settings, jumpers, etc).



- ♦ RS 232C/RS 485 cable connections at both the ends. In case of RS 485 communication, make sure that the PC/CRT is able to communicate with RS 485 Standard. RS 485 communication interface card should be installed in the terminal.
- ♦ Proper connections of all signals of RS 232C/RS 485 interface (Ref. Appendix E)
- ♦ Baud rate configured on trainer should match with that of the serial terminal connected to it. Ensure this by setting Dip Switch positions appropriately. Also check for other serial parameters (Character length, Stop bits, etc)
- ♦ If a computer system is the controlling device, check that the driver program is running and the serial cable is connected to the correct and a working serial port.
- ♦ If an LCD is interfaced with the trainer and a sign-on message appears, modify the serial parameters as required and verify them by pressing RESET key every time a change is made.

NOTE: DIP Switch status is read only at Power ON / Reset. If you change the settings, RESET the trainer.

If the problem still persists, please contact the manufacturer.

For stand-alone mode operation:

1. Configure the trainer using DIP Switch and jumpers as described in Section 2.2
2. Connect a PC Keyboard to the DIN connector provided
3. Connect the power supply of required capacity to ESA 86/88-3 and Power ON the trainer.

Now if 8086 is installed, the following sign-on message will appear on the LCD. The sign-on message is followed by the command prompt, "." in the next line. In case of 8088 CPU the sign on message will change accordingly.

```

      ESA 86 MONITOR Vx.y
      KBD      P:86
      .

```

Now ESA 86/88-3 is ready for operation in the keyboard mode.

No response in stand alone mode:

If the trainer does not respond satisfactorily in stand-alone mode, check the following.

- ♦ If the LCD backlight is off, check the power supply connections.
- ♦ If the display is blank, or shows random pattern, check the configuration settings once again.
- ♦ If after a normal sign-on message, there is no response to PC keyboard entries, check for proper connection at the keyboard connector.

NOTE: DIP Switch is read only at Power ON / Reset. If you change the settings, RESET the trainer.

If the problem still persists, please contact the manufacturer.



CHAPTER 3

COMMUNICATION WITH A HOST PC

3.1 INTRODUCTION

ESA 86/88-3-trainer kit is supplied with Windows and DOS driver packages, which allow the user to establish communication between ESA 86/88-3 trainer in serial mode and a host PC through its asynchronous communication ports (COM1/COM2/COM3/COM4, etc.). Both the packages fully support the ESA 86/88-3 Serial Monitor commands including file upload and download facilities. HEX files generated by PC native and Cross-tools can also be downloaded to the trainer using these packages. Further, data from ESA 86/88-3 memory can also be uploaded to the host PC. The compatible serial cable for RS-232C communication is also included in the package.

3.2 INSTALLATION

- a) Configure ESA 86/88-3 for serial mode of operation and set the serial port of ESA 86/88-3 for 9600 Baud and No parity (Refer Chapter 2).
- b) Connect the ESA 86/88-3 trainer to Host PC over the serial port using the serial cable provided. (Refer to your system's Technical Manual for details regarding the signal definitions on COM ports (Refer appendix E of this manual for serial cable details)).

3.3 INSTALLATION OF WIN863

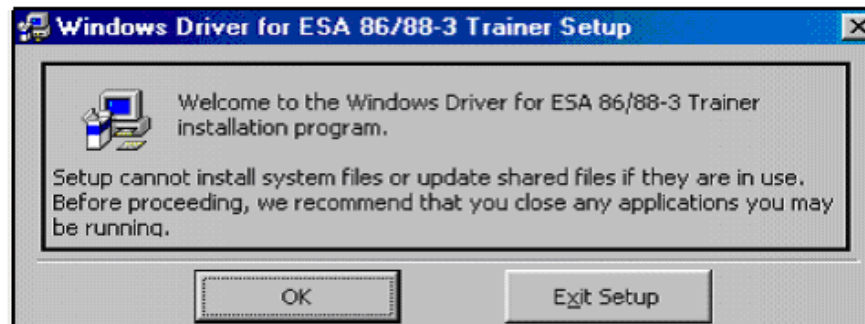
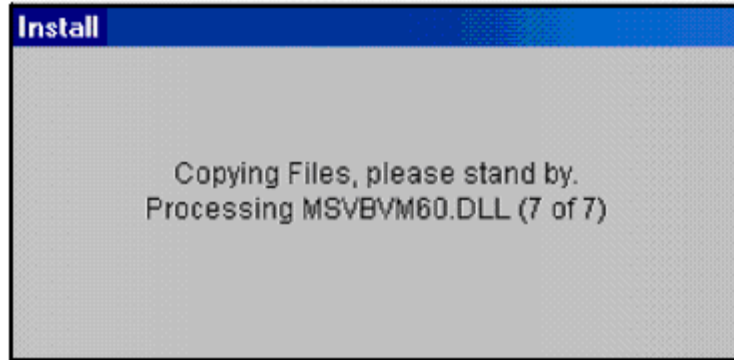
Win863 is a Windows based Communication package for ESA 86/88-3 trainer that provides a powerful and convenient debugging and development environment to the user. The user must install this software from the ESA 86/88-3 Drivers CD.

Insert the CD in the CDROM drive and run **Setup.exe** from the **Win863's** folder. The Setup program will guide the user through the rest of the installation procedure. Once the Software is installed successfully, Win863 offers a complete online help for working with the commands. Under Win863, the trainer can communicate at a baud rate of up to 38400.

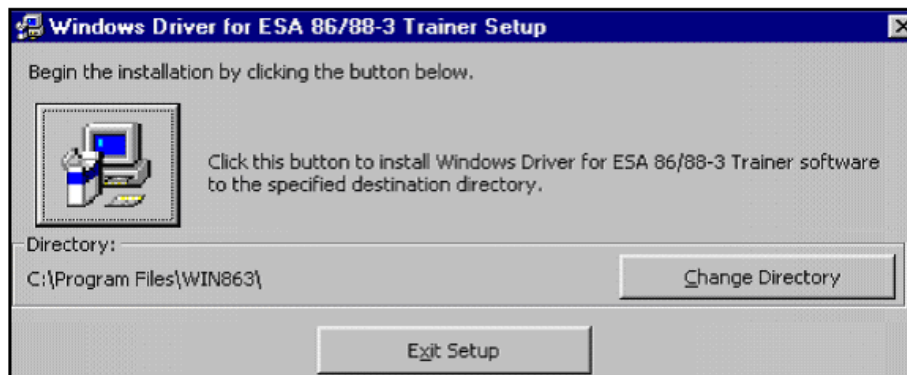
3.3.1 INSTALLATION PROCEDURE

- 1) Run **Setup.exe** from the Drivers Software CD. Now user can see the following dialog box, which will copy the required files from Source CD to the Host PC.





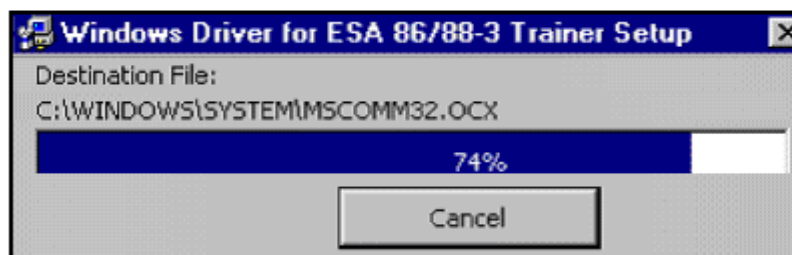
2) Click **OK**, then user can see the following dialog box.



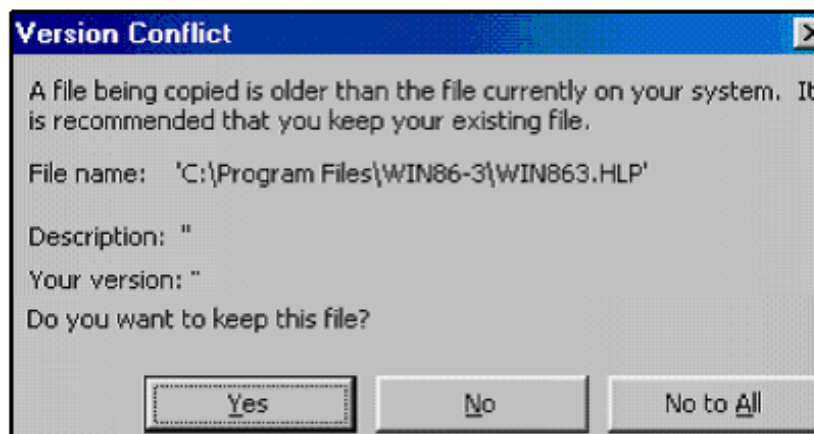
- 3) The default Installation directory will be C:\\ProgramFiles\\WIN863. User can change the installation directory by clicking **Change Directory**.
- 4) Click the highlighted Icon shown above to continue the Installation Procedure and the following Dialog Box.



5) Select the **Program Group**, Default **ESA Trainers**. Click **Continue**. Then user can see the following Dialog Box.



6) In some cases user will see the following Dialog Box, which will indicate the over writing of the existing files.



7) Click **Yes** to keep the old file. Click **No** to install the new file. Click **No to All** for the remaining files also.

8) Then user will see the following dialog box, which will indicate the completion of installation.

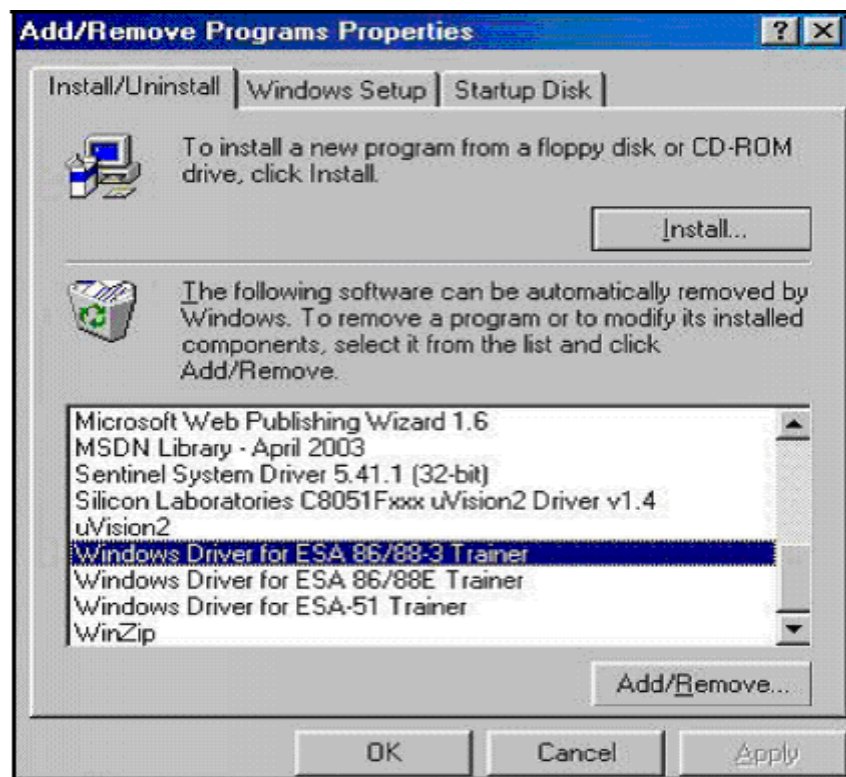




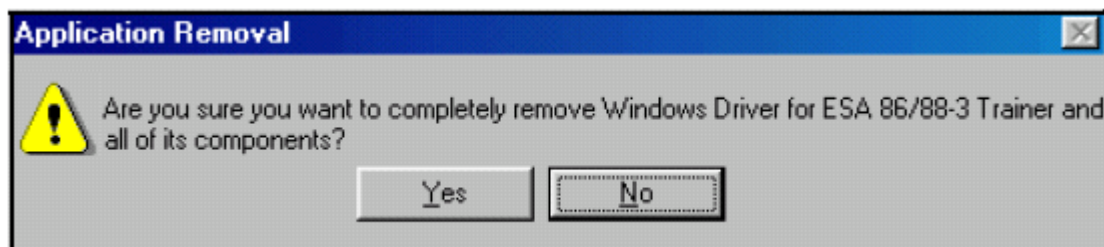
9) Click **OK** to complete the installation.

3.4 UNINSTALLATION PROCEDURE

1) Go to the **Control Panel** from **Start** Menu **Settings**. Select **Add/Remove Programs** option. User will see the following Dialog Box.



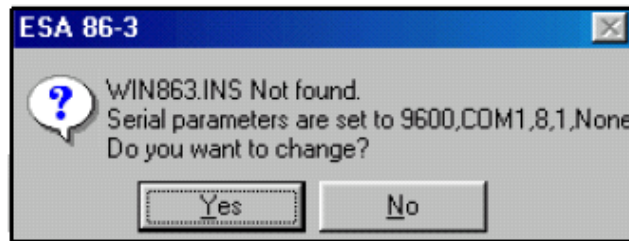
2) Select the Driver to Uninstall, and click **Add/Remove**. Then the following dialog box will appear, which will ask for confirmation.



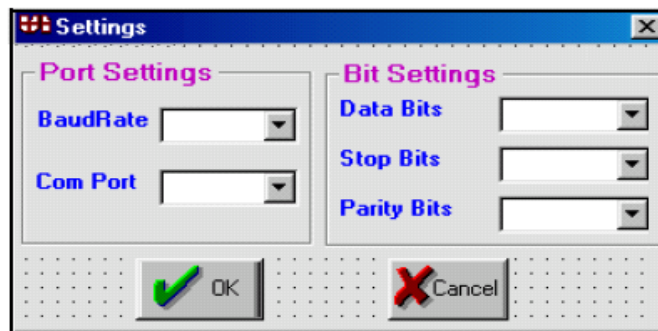
- 3) Click **Yes** to uninstall, otherwise **No** to come out from the uninstallation procedure.
- 4) Delete the installation Directory manually(C:\Program Files\WIN863).

3.5 CONFIGURATION OF ESA 86/88-3 TRAINER WITH WIN863

- 1) Switch on the PC.
- 2) Connect **ESA 86/88-3** Trainer to the Host PC using 9 to 9 pin serial cable supplied.
- 3) Connect power supply cable and switch on **ESA 86/88-3** Trainer.
- 4) Set the Dip Switches 3(9600 baud rate), 5(serial mode) and 8(RS-232C communication) in "ON" position on **ESA 86/88-3** Trainer, further information refer chapter 2.
- 5) Start **Win863** application on Host PC. Now user can see the following Dialog Box.

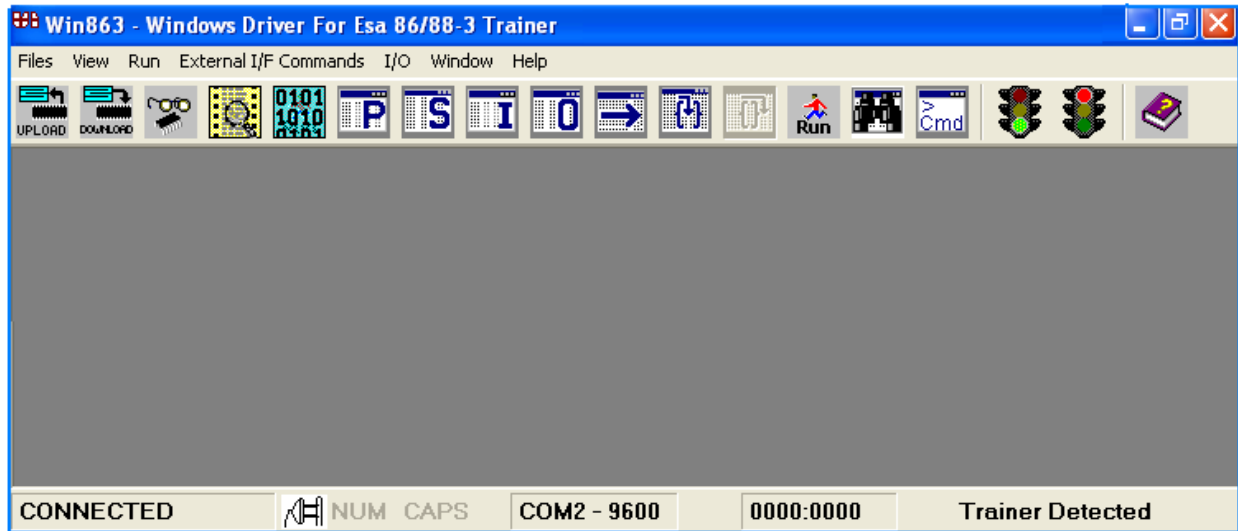


- 6) If the Trainer is connected to COM1 port on your Host PC & configured for 9600-baud rate, click **No**. Click **Yes** otherwise. If yes, then user will see the following dialog box.



- 7) Select the BaudRate as per the dipswitch configuration on **ESA 86/88-3** Trainer. Select the COM Port where the trainer is connected. Click Apply. Now user can see the following window.





COMMUNICATION STATUS

SIGN-ON MESSAGE

8) In some cases if the communication is not established you will see the following dialog box. In this case please check the ESA **86/88-3** Trainer is powered on. Check whether serial cable is connected to the PC & **ESA 86/88-3** Trainer and check the parameters by clicking the Settings.



9) If the communication is established properly, from the displayed window you can work with **ESA 86/88-3** Trainer. User can view the Registers, Memory Dumps, Downloading the hex images, Uploading the hex dump to the Host PC, Running the downloaded hex images, single stepping, breakpoints, watch windows, Memory modification etc.,

3.6 OPERATION DETAILS

The complete command set of the ESA 86/88-3 is transparent and is fully supported by Win863 (Refer chapter 4 for the serial monitor mode commands). Click on help icon in the Win863 dialog box for help. In addition, Win863 supports the file download, file upload and other commands, which are explained below.

NOTE: During parameter entry, the system expects the alphabetic characters to be in upper case.

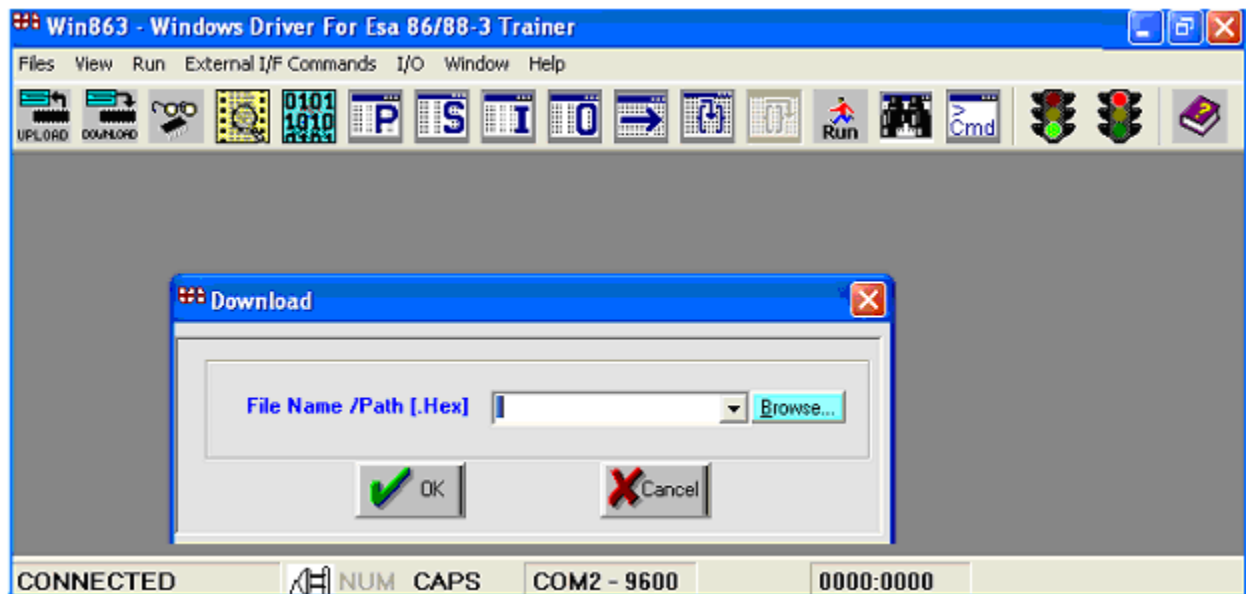


However, Win863 driver is not case sensitive. So, Keyboard with CAPSLOCK on is not required.

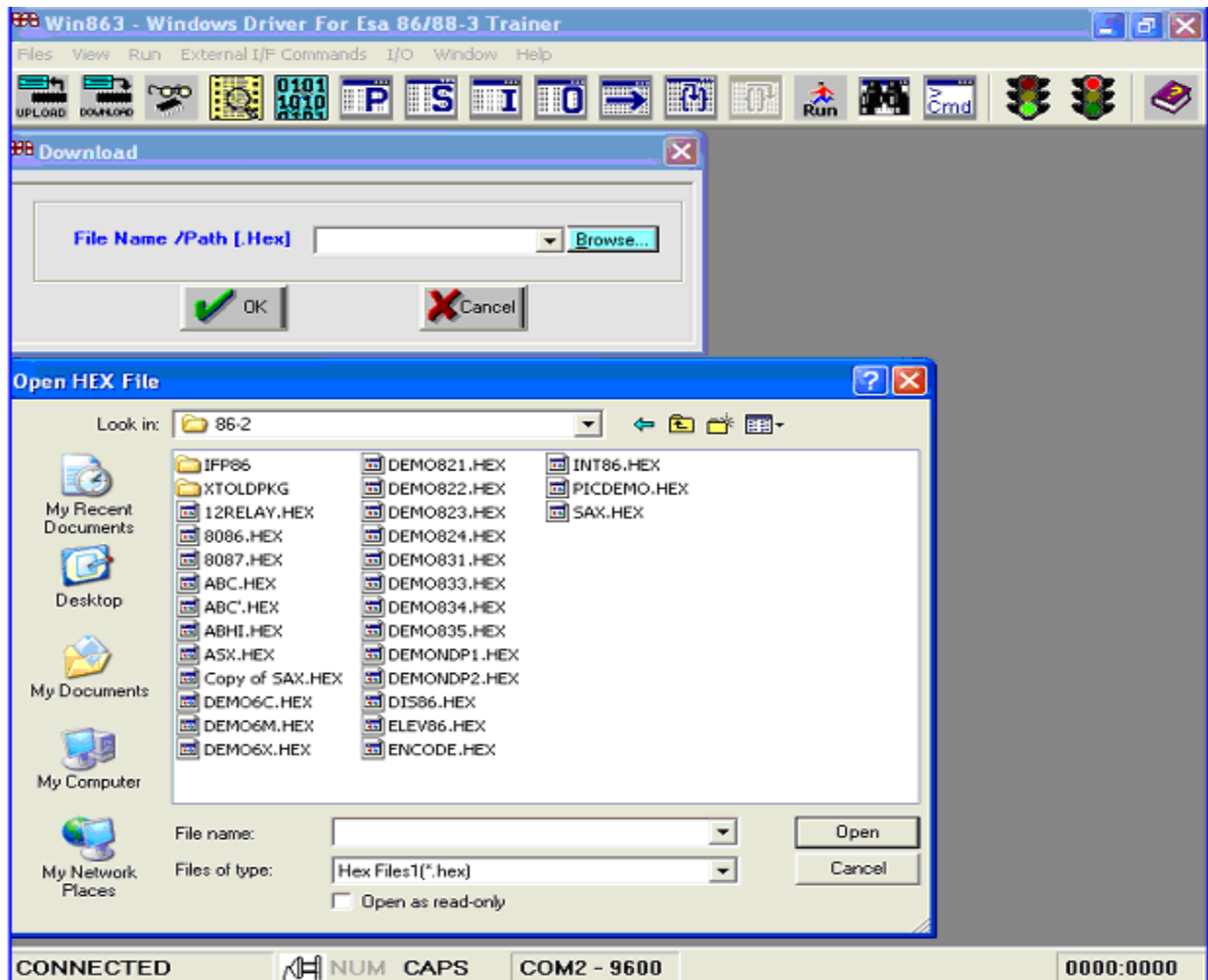
3.6.1 DOWNLOAD OPERATION

This feature allows to download the contents of an object code file into the memory of ESA 86/88-3. The object code file must be a ".HEX" file with records in INTEL 8-Bit HEX format. Please refer to the relevant INTEL manuals for the definition of INTEL 8-Bit HEX format. Most of the cross assemblers for 8086 do produce object code files which are ".HEX" files with records in INTEL 8-Bit HEX format.

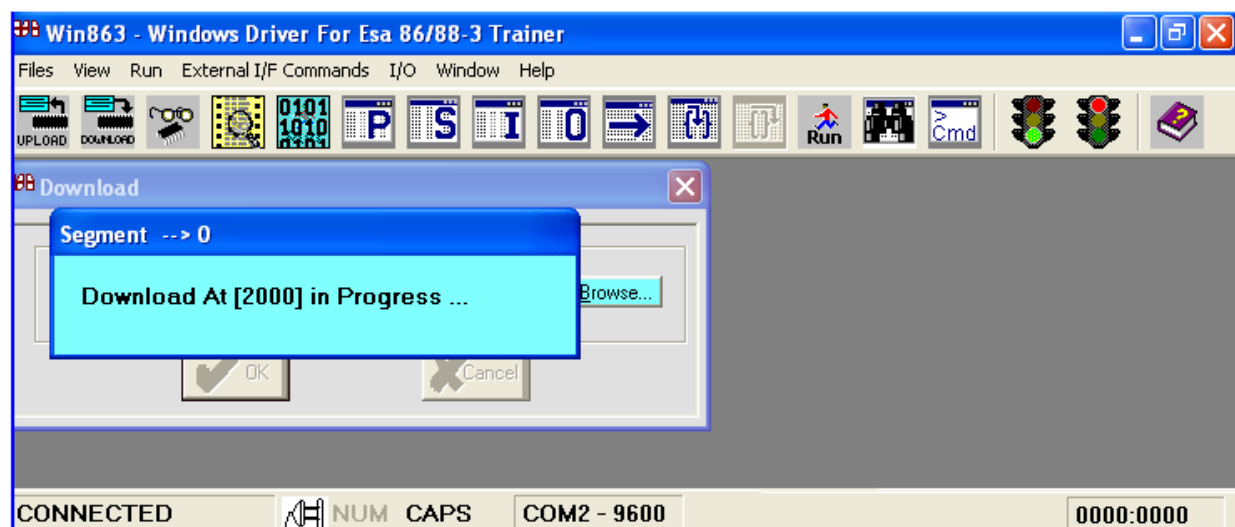
1) To perform download operation click on download button or go to file menu and choose download option in the Win863 dialog box. Now user can see the following dialog box.



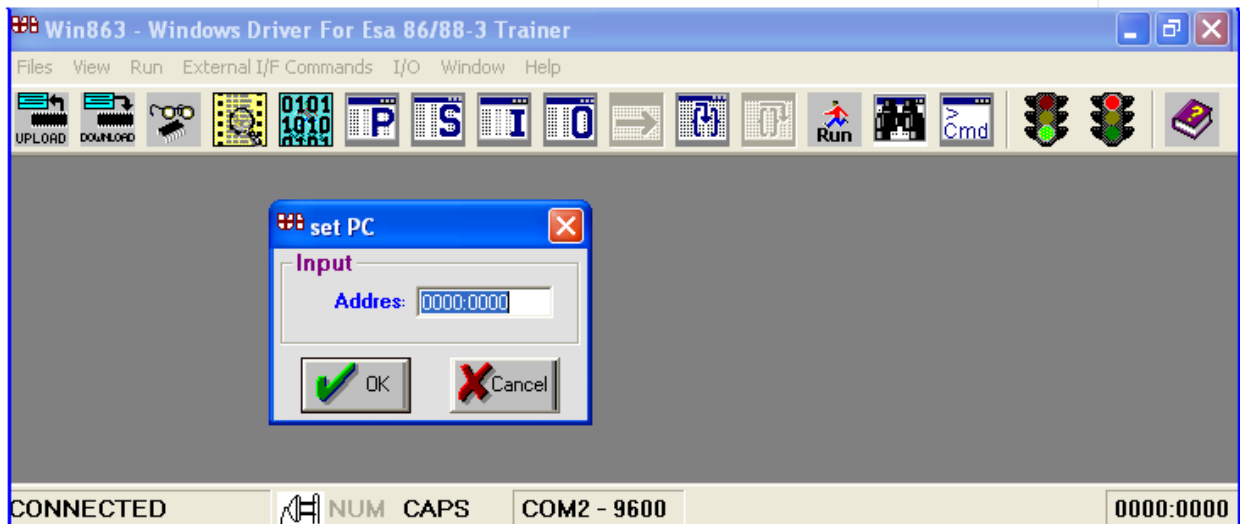
2) Here click on "**Browse**" tab and select hex file name which user wants to download. Then user can see the following dialog box



3) Select the hex file name from the open HEX file window and click on OK in the download dialog box. Then user can see the following dialog box

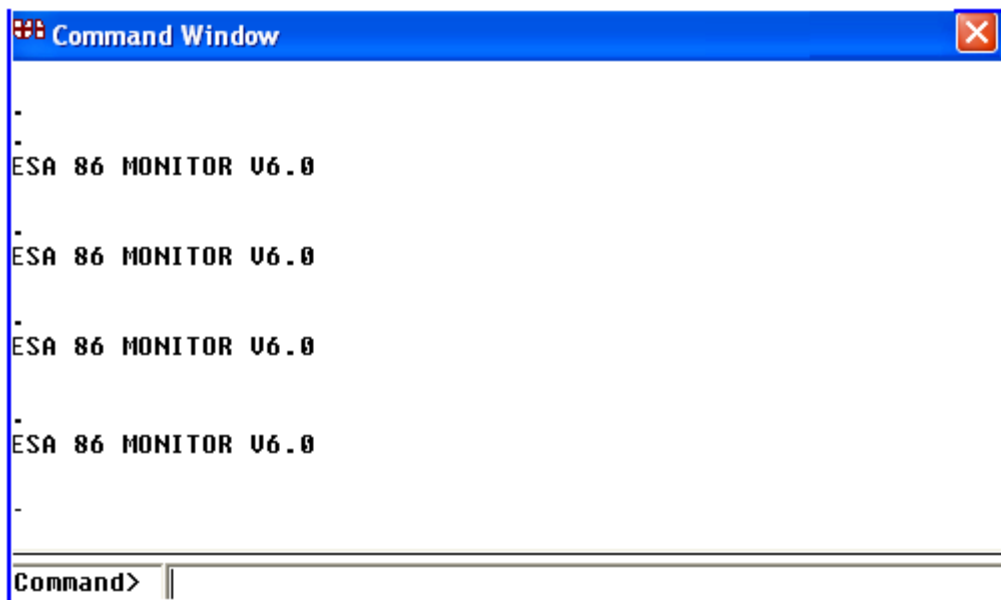


5) After downloading the program user can execute the program without going to command window also. For that user has to go RUN menu and select SET PC option then see the following dialog box



6) In that Set PC dialog box user can give the program starting address and click on OK. Now go to RUN menu and select RUN, then the program will execute and user can see the output.

7) Otherwise go to view menu and select command window option, then user can find the following dialog box



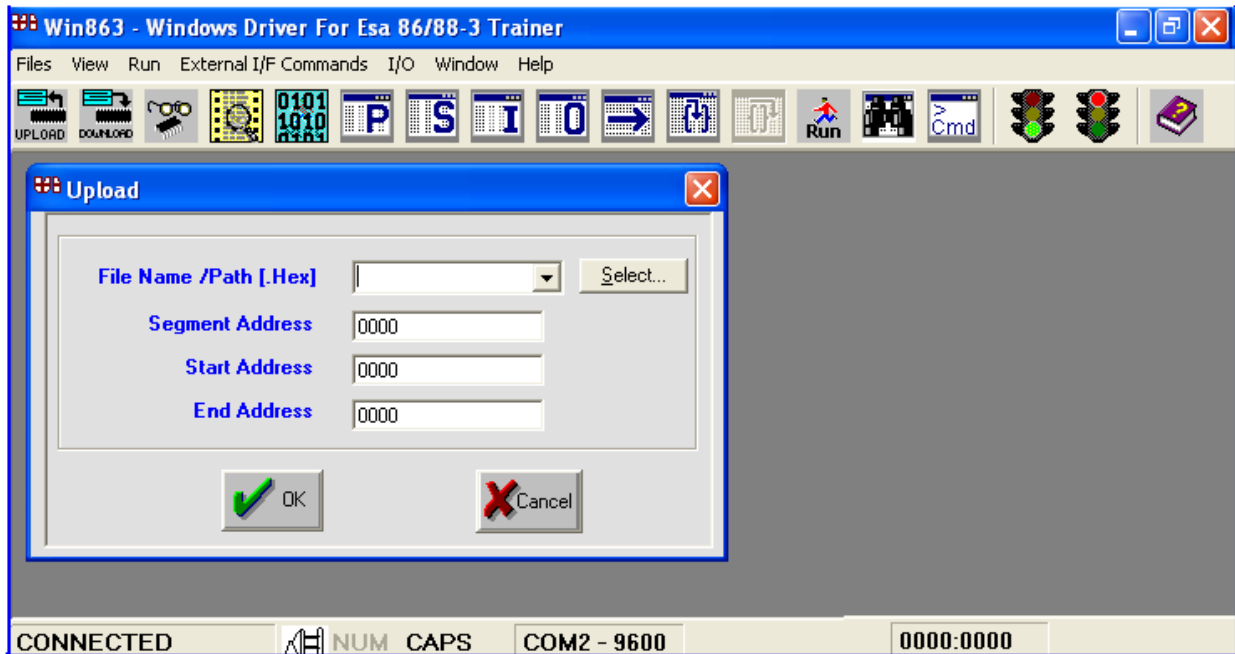
Here user can give GO command and execute the program.



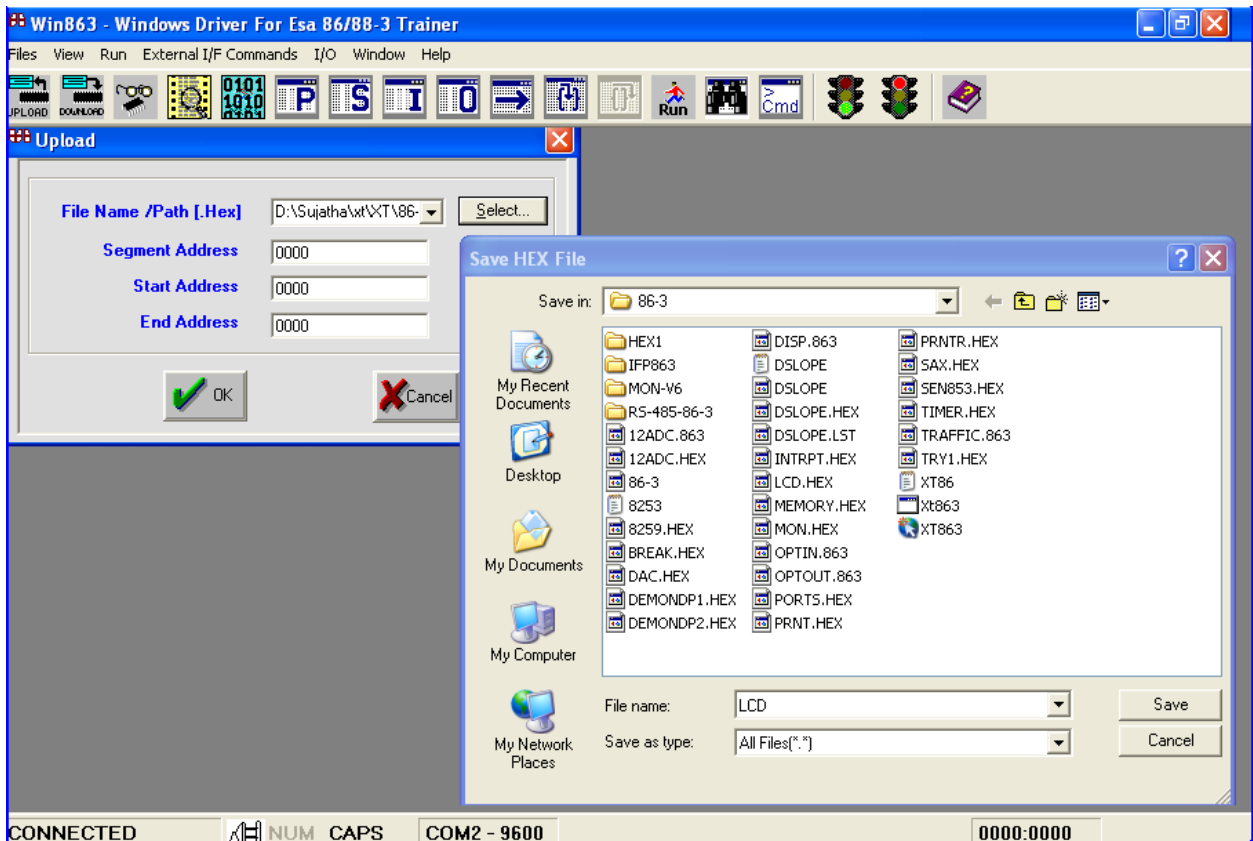
3.6.2 UPLOAD OPERATION

This feature allows uploading of the data from the memory of ESA 86/88-3 to the computer and saves the data in the specified disk file in INTEL 8-Bit HEX format.

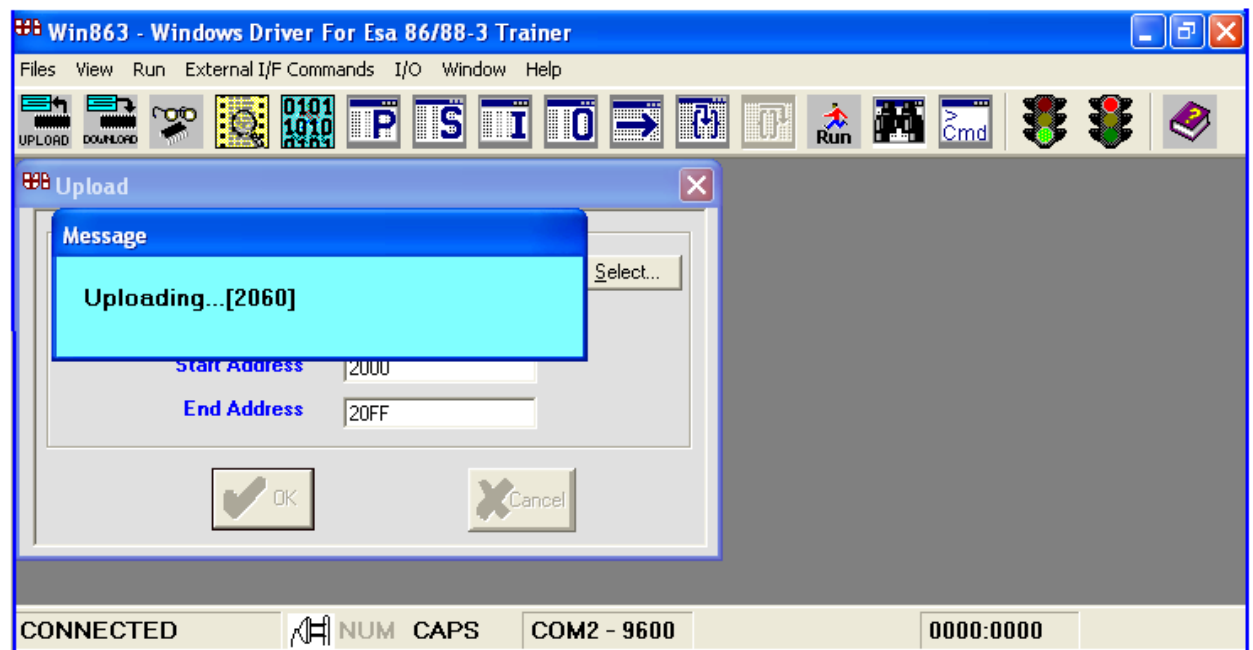
- 1) To perform download operation click on upload button or go to file menu and choose upload option in the Win863 dialog box. Now user can see the following dialog box.



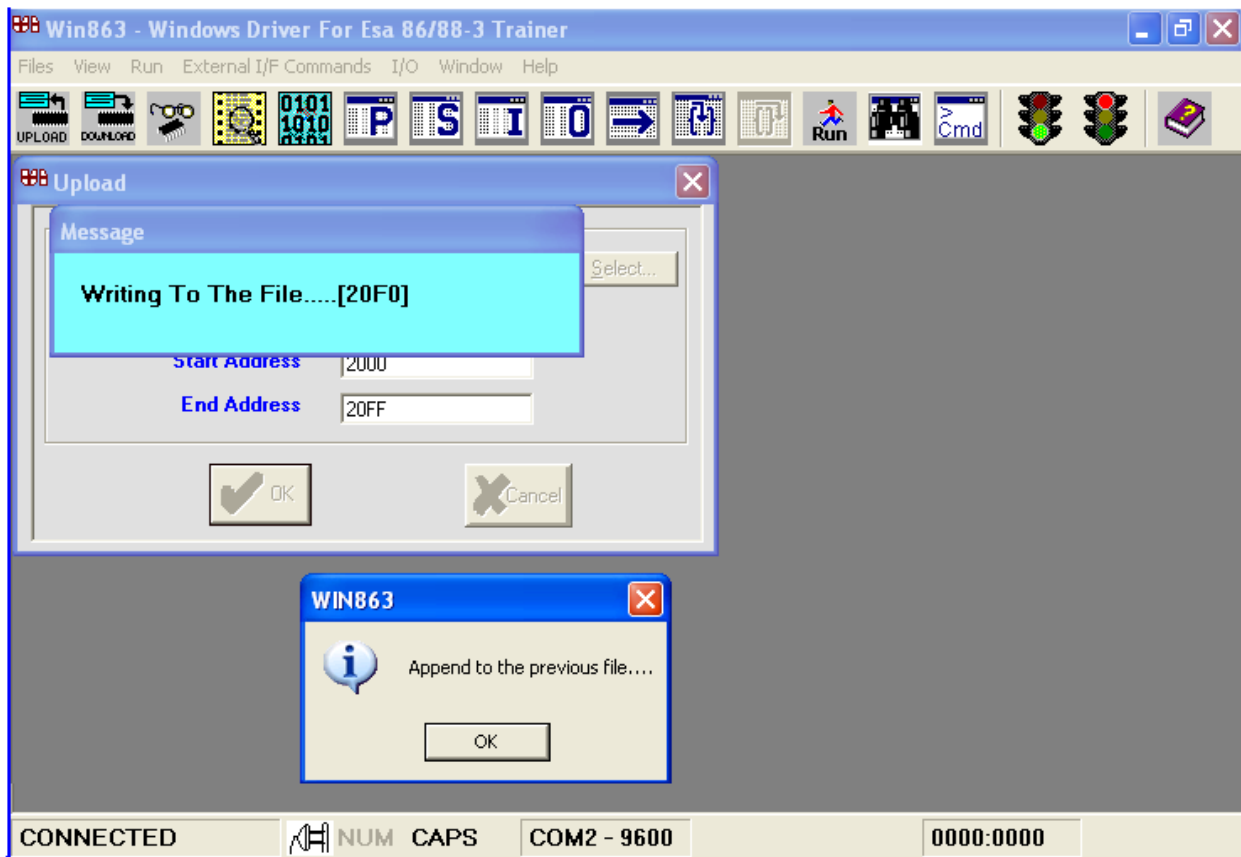
- 2) Here click on **"Select"** tab and give the name the file to save then user can find following dialog box. Here user can select the path where the upload file should be.



3) After saving the file enter the starting and ending addresses of the file which user is going to upload and click on OK of the upload dialog box as follows



4) click on CANCEL in the upload dialog box. If user wants to add one more file to same uploaded file once again click on OK in the upload dialog box. Then user can find the message as follows



5) click on OK to append the file to previous file and click on CANCEL. After this operation user can download the upload file as already mentioned above in the 3.6.1 session.

The following error messages may appear during upload and download operations.

1. File not found !
2. Path not found !
3. No more files !
4. Access denied !
5. Invalid file handle !
6. Insufficient Disk space !
7. Unable to continue upload !
8. Colon is not present at the start of the Record !
9. Invalid data in (source file) the following Record !
10. Checksum Error in the following Record !

3.6.3 COMMUNICATION



Communication parameters can be set during the session by pressing Ctrl+O or go to view menu and select options in the Win863 window. List of parameters and their current values will appear on the dialog box. Select the desired parameter with the help of arrow keys. The parameters allowed to be set are Communication Port (suitable COM), Baud Rate (110 / 150 / 300/ 600 / 1200 / 2400 /4800 / 9600), Number of Data bits (7/8), Number of Stop bits (1/2) and Parity (NONE/ODD/EVEN). After selecting the desired values click Ok in the following dialog box. Otherwise click on CANCEL to ignore the values.

Table 3.1 shows details of the integer values and corresponding parameters.

Commn. Port	int1	Baud Rate	int2	Data Bits	int 3	Stop Bits	int 4	Parity	int
COM1	0	110	0	7	0	1	0	odd	0
COM2	1	150	1	8	1	2	1	none	1
COM3		300	2					even	2
COM4		600	3						
.		1200	4						
.		2400	5						
.		4800	6						
.		9600	7						

Table 3.1

3.6.4 HELP

On-line help is available for all ESA 86/88-3 monitor commands and specific commands of

WIN863. Help facility can be selected by selecting help option in the window. A menu of commands is displayed from which desired command can be selected and help information about that command is displayed. These options are explained in detail in online help of **WIN863 (WIN863.hlp)**.

Note: Whenever the command window is full, it is recommended to clear the command window with " CLS " command. Otherwise display will become a little bit slow.

TROUBLE SHOOTING:

- 1) Check the PC Serial Port working condition.
- 2) Check the Serial Cable working condition.
- 3) Check **ESA 86/88-3** Trainer Kit for Sign on message
- 4) Dipswitch Settings

3.7 RS-485 INTERFACE

ESA 86/88-3 has optional provision for serial communication using RS-485 standard also. However it is essential that the host PC must be capable of communicating with this



communication standard. The user may obtain ESA 422/485 serial card and cables from ESA as an optional hardware support for RS-485 communication. The driver software can also be obtained from ESA for communicating with ESA 86/88-3. Please contact the manufacturer for further details in this regard. For dipswitch settings on ESA86/88-3 trainer refer chapter 2.

3.8 MORE ABOUT “.HEX” FILES

The files uploaded/downloaded under the control of Win863 are in the INTEL Extended Hex format, which is described below:

3.8.1 Intel Extended HEX Format

In this format, binary values are coded in ASCII. For example, the binary value 0100 1111(=4FH) is coded as two ASCII characters “4” AND “F”. In other words, the byte 4FH is represented by two bytes 34H (4 in ASCII) and 46H (F in ASCII). Thus this representation requires twice as many bytes as required by binary representation. A file in extended Hex format can consist of up to four types for records. Each record begins with a record mark field containing 34H, the ASCII code for colon (:). This is followed by a RECORD LENGTH field, which specifies the number of bytes of information, which follow the RECORD TYPE field. The maximum number of record bytes allowed in the record is 255. After the data, the record has a CHECKSUM field that contains the ASCII representation of the two’s complement of the eight-bit sum of the bytes that result from converting each pair of ASCII hexadecimal digits to one byte of binary, from and including data field. Therefore, the sum of all the ASCII pairs in a record after converting to binary, from the RECORD LENGTH field to and including the CHECKSUM field is zero. The record ends with carriage Return, Line Feed sequence (0DH, 0AH).

The four types of records that may be present in a file of Extended Hex format are:

- Extended Address Record
- Start Address Record
- Data Record
- End of File Record

Extended Address Record:

RECD	REC	ZERO	REC	USBA	CHECK
MARK	LEN		TYPE		SUM

```
“:”      “02”  “0000” “02’  XXXX  XX
```

This record is used to specify the bits 19-4 of the Segment Base Address (SBA) where bits 3-0 are zero. Bits 19-4 of the SBA are referred to as the Upper Segment Base Address (USBA). Subsequent data bytes are loaded at the specified offsets relative to this USBA. The Extended Address Record may appear anywhere within the file. This value remains in effect until another Extended Address Record is encountered.

Example Record:

```
:02  0000  02    0200  FA
```

02+0000+02+0200+FA = 00 checksum validity



USBA = 0200H. For example, third byte in a subsequent data record with a load address of 0100H is loaded at 0200:0102.

Data Record:

RECE		REC	LOAD	REC	_.DATA._	CHECK
MARK	LEN	ADDRESS		TYPE		SUM
":		XX	XXXX	"00"		XX

The data record provides a set of ASCII characters that represent the hex digits of the data bytes. The method of loading these data bytes has already been described in the discussion of the Extended Address Record.

Example Record:

: 06 00 10 00 BA FF FF B0 40 90 B2

06+00+10+00+BA+FF+FF+B0+40+90+B2=00 checksum validity

Load address = 0010 & Number of data bytes =06

Assume the USBA currently valid is 0200. Then the data is to be loaded as shown below:

0200:0010=BA 0200:0011=FF 0200:0012=FF
0200:0013=B0 0200:0014=40 0200:0015=90

Start Address Record:

RECE		REC	LOAD	REC	CS	IP	CHECK
MARK	LEN	ADDRESS		TYPE			SUM
":		"04"	"0000"		"03"	XXXX XXXX	XX

The start address record is used to specify the execution start address for the object file. This record can appear anywhere in the .HEX file. If the start address record is not present, CS and IP values will remain what they were before downloading.

Example Record:

: 04 00 00 03 02 00 01 00 F6

04 +00+ 00+ 03 +02 +00 +01 +00+ F6 = 00 checksum validity.

After loading, CS =0200 and IP= 0100

End of File Record:

RECD		REC	ZERO	REC	CHECK
MARK	LEN		TYPE	SUM	
":		"00"	"0000"	"01"	"FF"

This is a record of fixed structure and is used to specify the end of the .HEX file.

3.9 USING A CROSS ASSEMBLER



A convenient way of creating a file to be downloaded into ESA 86/88-3 is to use a cross-assembler for 8086/8088 that can generate the object code in Extended Hex format.

Example: Chapter 10 includes some programs that have been developed using X8086 cross assembler. The distribution CD-ROM contains HEX files (object code) for these examples. The code is in Extended Hex Format and the files were created using X8086 and the corresponding linker LINK. The user can download these files using the download command and execute them according to previously given guidelines

The steps involved in creating a .HEX file are:

1. Create a source file using the DOS text editor or notepad and save it as **filename.ASM**.
2. Assemble **filename.ASM** using X8086 to create **filename.OBJ**.
3. Link the single file **filename.OBJ**.

Specify code offset according to the memory organized in the trainer. You may even skip this prompt if "ORG" directive has been used in the source file. Select E (for Extended Hex) as the option for output file format. This process creates **filename.HEX** that can then be downloaded to ESA 86/88-3 trainer through Win863.

Note: The execution start address record generated by X8086 Linker does not follow the standard described already and hence is not supported by Win863. Thus user should not utilize this feature of X8086. In other words, the END statement in the source program should not have any operands.

Downloading .EXE files

If the source program in assembly language is assembled using the Macro Assembler MASM and then linked using the Microsoft Linker, the resulting object code file will have .EXE file structure. Such an .EXE file has two parts:

- Control and relocation information
- The load module itself.

This structure is designed for convenience in executing the files on a PC compatible host PC under PC DOS/MS DOS/WINDOWS operating system (Please refer to the Technical Reference Manual of DOS for complete details regarding .EXE file structures). Thus to get absolute object code from .EXE files, one has to determine the load segment from where the load module is to be located and then relocate it appropriately using the control and relocation information present in the .EXE file. This process is accomplished by the LOADER program on the computer system. Because of these factors, downloading .EXE files into ESA 86/88-3 (an altogether different execution environment) is not a convenient procedure and user is encouraged to utilize X8086 cross-assembler. However .EXE files with total size less than 64K can be downloaded as described in the following steps:

a) Convert the .EXE file to HEX file using the file converter package EXE2HEX.COM. This file is included on the distribution diskette. To invoke the file converter, type EXE2HEX



followed by RETURN at the DOS prompt. The system will now display the following sign-on message:

ELECTRO SYSTEMS ASSOCIATES PVT LTD
BANGALORE
EXE TO HEX FILE CONVERTER V1.0

Then the system will prompt for input file name as shown below:

FILE NAME =

Enter the file name with the extension followed by RETURN. If the specified filename is an invalid one (refer to your DOS manual for file name conventions), it displays the message "WHAT FILE SIR!" and returns to DOS. If the specified file does not exist, it displays the message "FILE NOT FOUND!" or the message "PATH NOT FOUND!" and returns to DOS. If the file can not be read, it displays the message "ACCESS DENIED!" and returns to DOS. If a file already exists on the system with the same name as the input file name but with the extension of HEX, then it displays the message "HEX FILE ALREADY EXISTS!" and returns to DOS. If none of these errors occur, it creates an output file with the extension of .HEX. If it fails in this process, it will display the error message "TOO MANY OPEN FILES!" and returns to DOS. Otherwise the system will prompt for the LOAD SEGMENT value as shown below.

LOAD SEGMENT =

Enter the desired load segment value followed by Return. The system will now prompt

ORIGIN VALUE =

Enter the origin value by RETURN.

NOTE: The advantage of specifying an origin value is as follows:

However if user wants to load from origin 0 (for example with a load segment value of 200H), user can specify the ORIGIN VALUE to be 0. Thus this feature allows user to delete if required, unwanted data from the HEX file. Now the system will read the .EXE file and check if it is a valid .EXE file. If the specified input file does not have the .EXE file structure, the system displays the message "IMPROPER .EXE FILE!" and returns to DOS. If it is a valid .EXE file, then it will be relocated using the specified load segment value and the origin value, in accordance with the control and relocation information present in the .EXE file. The system will create an Extended Address Record using the load segment value and write this record on the output file. During relocation, if the system detects that the file size is greater than 64K, it displays the message "FILE TOO LARGE FOR CONVERSION!" and returns to DOS. If the relocation process is completed successfully, the system will write the relocated information on the output file as records in HEX format. Then it will obtain the CS and IP values from the .EXE file, relocate them and then create a start address record using the relocated CS and IP values. Then an end address record is generated and written to the output file. If the system fails in



writing to the disk file, it displays the message "DISK WRITE FAILURE" and returns to DOS. After completing the process of writing, it closes the .HEX file. During the entire process, the hex records being created are displayed on the screen also. The system will then extract SS and SP values from the .EXE file, relocate them and then display them on the console. If required, these values can later be used to set the SS and SP registers separately. Please note that DS and ES registers are supposed to point to an area called Program Segment Prefix when the .EXE file is given control under MS-DOS. However, when the conversion to HEX structure is performed for downloading the information into ESA 86/88-3, the Program Segment Prefix has no relevance and thus the system ignores these values.

b) After converting an .EXE file into a .HEX file as described above, user can install and run the Win862 and download the .HEX file into the memory of ESA 86/88-3 as per the procedures already described.

c) After downloading the file, user must set the DS, ES, SS and SP registers to the required values using the X command of the Serial Monitor of ESA 86/88-3. The SS and SP values displayed by EXE2HEX.COM can be made use of, if required.

d) Before executing the downloaded program, user is strongly urged to disassemble the code (or at least important parts of the code) using the on-board disassembler of ESA 86/88-3 and ensure that the program is correct and that the segment registers are set to proper values.

e) Execute the downloaded program.

Downloading .COM Files

Assume that an .EXE file has been created using MASM and LINK as described in the previous section. Such a file can be converted to a .COM file using the standard DOS command EXE2BIN (Refer the DOS user's manual for details regarding this command and for limitations on the structure of .EXE file which are to be converted to .COM files).

A. COM program contains the pure memory of the program and has the following distinct characteristics:

- a) The size of the code and data parts of the file is less than 64K.
- b) There is no separate STACK segment
- c) No segment fixup segment relocatable.
- d) CS, DS, ES and SS are assumed to contain the same value when the program begins execution.

This structure is designed for executing small programs, in a convenient way on PC/XT/AT compatible computer systems, under PC DOS/MS DOS. However, a .COM file can be downloaded into ESA 86/88-3 as described below:



a) Convert the .COM file to .HEX file using the file converter package COM2HEX.COM which is included on the distribution CDROM. To invoke the file converter, type COM2HEX followed by RETURN at the DOS prompt. The system will now display the following sign-on message:

ELECTRO SYSTEMS ASSOCIATES PVT LTD

BANGALORE

COM TO HEX FILE CONVERTER V1.0

Then the system will prompt for the input file name, load segment value and origin value in a manner exactly similar to the one described in the previous section. The meaning and interpretation of these parameters and the possible error messages are also same as in the previous section. However, as a .COM file is not expected to have a separate stack segment COM2HEX does not display SS and SP values as is done by EXE2HEX program.



CHAPTER 4

ESA 86/88-3 MONITOR COMMANDS

ESA 86/88-3 Microprocessor can be operated either in serial or stand-alone mode. This chapter describes the serial and standalone mode monitor commands used for working with the trainer. Most of the monitor commands in either mode of operation bear uniformity in syntax and operation. The following discussion on Monitor commands is therefore common to either mode of trainer operation, unless otherwise specified.

4.1 GENERAL OPERATION

Installation and Configuration of ESA 86/88-3 is described in Chapter 2. It is essential that the trainer should be configured properly for working with it smoothly. Getting a proper sign-on message upon RESET as explained in the previous chapter is the minimum requirement for working with ESA 86/88-3

Reset Status: On Power ON/Reset, all information about the previous user program is lost and the registers may acquire new data. However the contents of user RAM are not disturbed if onboard RESET is used. If the RAM is backed up with battery, then the user RAM data is not lost even if power is switched OFF.

Resetting the trainer initializes the segment & status registers of the CPU as described below.

Table 4.1 Register Initialization

Register	Value (HEX)
CS (Code Segment)	0
DS (Data Segment)	0
ES (Extra Segment)	0
SS (Stack Segment)	0
IP (Instruction Pointer)	0
FL (Flags)	0
SP (Stack Pointer)	100

Further, interrupt vectors 1, 2 and 3 are initialized as follows:

Interrupt 1: Single Step Interrupt -This interrupt is used by the monitor with the step command.

Interrupt 2: NMI (Non-maskable Interrupt)- This interrupt is implemented by BREAK key and the vectoring information is completely user defined. .

Interrupt 3: Breakpoint Interrupt-This interrupt is used with GO command and its use is at the user's discretion.

A detailed discussion of the 8086/8088-interrupt vectors can be found in the Intel's 8086 Family User's Manual (Chapter 2, Processor Control and Monitoring, Interrupts PP 2.22-2.28).



Whenever the monitor is re-entered as a result of a single step or Breakpoint Interrupt, the monitor saves the contents of the 8086/8088 registers on the User Stack and subsequently restores the register contents from the stack before it prompts for command entry. Since the SP register is initialized to 0100H, and memory locations 0H-CFH are reserved for monitor, the stack length reserved for the user is 48 bytes (i.e. locations D0H - FFH). Of these, 26 bytes must be left for register contents, should one of the above interrupts occur (leaving 22 bytes for the user). In addition locations 100H to 1FFFFH are used for other system functions and system tables. Hence the user RAM starts from 0:2000H

4.2 STRUCTURE OF MONITOR COMMANDS

When the monitor is ready to accept a command from the user it outputs a period ('.') as the command prompt character at the beginning of a new line.

The commands entered by the user consist of one or two-character command mnemonic followed by a list of command parameters. This list may be up to three parameters long depending on the command being used. When more than one parameter is required, a single comma (',') is used between the parameters as a separator. A command is terminated either by a carriage return or by Escape depending on the command itself. Commands are executed one at a time and only one command is allowed in a command line.

4.2.1 RULES OF PARAMETER ENTRY

All addresses in 8086/8088 systems consist of a segment value and an offset value. The segment value is entered first, a colon (':') is entered as a separator and then the offset value is entered. If the segment value is not specified (note that in such a case, the colon also should be omitted), the default segment value is the current content of the code segment (CS) register. The address is entered as a hexadecimal value, most significant character first. The valid range of hexadecimal values for an address entry (either segment or offset) ranges from 0000 to FFFFH. If more than four digits are entered, only the last four digits entered are valid. In other words all address values are interpreted modulo 64K.

Data is also entered as hexadecimal value, most significant character first. The valid range for data entries is 00 to FFH for byte entries and from 0000 to FFFFH for word entries. If more than two (for byte entries) or four (for word entries) digits are entered, only the last two or four digits entered are valid.

Examine/Modify register command (X) requires register symbols as parameters. The register abbreviation entries required by the X command are described in detail with the explanation of this command.

4.2.2 USE OF +/- OPERATORS AND REGISTERS, IN PARAMETERS SPECIFICATION

While the address/data parameters are to be entered as hexadecimal values, the operators '+' and '-' can be used to form expressions and also to specify the use of the contents of any 8086/8088 register as address/data values.



For example, suppose we wish to specify an address whose segment is (ES) + 10H and whose offset is (BX) - 30H. One way is to calculate these values as shown below.

Assume (ES) = 0270H and (BX) = 0080H
Segment value = (ES) + 10H = 0280H
Offset value = (BX) - 30H = 0050H

Then the address can be specified as 0280:0050

However, ESA 86/88-3 supports direct specification of such an address value as shown below:

ES + 10:BX - 30

The system will automatically evaluate such expressions and use the correct values. These operators can be used to considerably simplify the specification and checking of relative addresses also.

4.2.3 RESPONSE TO ERRORS

Whenever an error is detected the command is aborted, the symbol ('?') is output on the command line, a carriage return and line-feed are issued and the command prompt character (`. `) is output at the beginning of the new line.

Command execution occurs only after a command terminator (a comma or a carriage return, depending on the command) is entered. Hence a command entry can be cancelled any time before the terminator is entered. The monitor detects entry of any character that is not legal or does not match the expected entry. The monitor then aborts the command, displays a '?' and returns to command entry mode.

4.3 ESA 86/88-3 MONITOR COMMANDS

The Monitor commands in both serial and stand-alone mode are similar and the following discussion holds good for trainer operation in either mode. These commands are summarized in the following table and are described in detail in later sections.

In both the table and individual command descriptions, the following notation is used.

[V] indicates that 'V' is optional
[V]* indicates one or more optional occurrences of "V"
<V> indicates that 'V' is a parameter to be entered by the user.
,/- indicates that either of the two characters is to be entered.

NOTE: These symbols are used to clarify the command formats and they are neither to be entered by the user nor output by the system.

In the description of the individual commands, it is assumed that 8086 CPU has been installed. However, all the commands work in identical fashion even if 8088 CPU is installed. The only observable difference would be in the sign-on message generated by the system following Power ON/Reset.



4.3.1 SUMMARY OF ESA 86/88-3 MONITOR COMMANDS

COMMAND	FUNCTION	FORMAT / SYNTAX
S	Substitute Memory bytes: Displays / modifies memory bytes	S [<address>] <CR> [,<new data>] ,] * <CR>
SW	Substitute Memory Words: Displays / modifies memory words.	SW [<address>]<CR> [,<new data>] ,] * <CR>
D	Display Memory bytes: Displays block of memory in byte format	D <start address> [,<end address>]<CR>
DW	Display Memory Words: Displays block of memory in word format	DW <start address> [,<end address>] <CR>
X	Examine / modify Registers: Displays modifies 8086/8088 CPU registers	X [<reg><CR> [<new data>/,]] <CR>
M	Move Memory: Copies a block of memory from one location to the other.	M <start address> , <end address>,<destination address> <CR>
F	Fill Memory (Bytes): Fills a block of Memory with constant byte data	F <start address>,<end address>,<byte value><CR>
FW	Fill Memory (Words): Fills a block of Memory with constant Word data	F <start address>,<end address>,<byte value><CR>
I	Input byte: Accepts and displays the data byte at the input port	I <port address> <CR>[,] * <CR>
IW	Input Word: Accepts and displays the data word at the input port	IW <port address> <CR>[,] * <CR>
O	Output byte: Outputs a data byte to the output port	O <port address><CR>[<data>/,]* <CR>
OW	Output Word: Outputs a data word to the output port	OW <port address> <CR>[<data>/,]* <CR>
C	Compare Memory: Compares a block of memory with another block	C <start address1>,<end address1>,<start address2><CR>
G	Go (Execute): Transfers the processor control from the monitor to the user program address with optional breakpoint.	G <CR> [<start address> [,<breakpoint address>]] <CR>
N	Single Step: Executes single instruction of the user program	N <CR> [<start address>],[,<new address>]* <CR>
H	Help Command: Lists Monitor commands with their valid syntax.	H [<Command mnemonic>]<CR>



P*	Invoke Programmer Software: Invokes the software for ESA EPROM Programmer Interface	P<CR>
A**	Enter Assembler: Invokes ESA 86/88-3 Symbolic One-line Assembler	A [address]<CR>
LL**	List Labels: Lists all labels defined in the Symbol table	LL<CR>
LC**	Label Clear: Clears all previously defined labels from the Symbol Table	LC<CR>
Z**	Disassembly Command: Disassemble Hex Code into 8086 mnemonics for a specific memory range	Z[<start address>[,<end address>]]<CR>

** Refer Chapter 7 for a detailed discussion of this command*

*** Refer Chapter 8 for a detailed discussion of these commands.*

4.4 COMMAND OUTPUT CONTROL

1. In Serial Mode:

During Serial operation, the output to the console can be stopped using the following method. This is applicable in cases of commands wherein the monitor continuously outputs data to the terminal viz. Display memory, Disassembly and compare memory commands.

Control-C, entered at any time, immediately terminates the command and the monitor returns to the command entry mode. Entering **Control-S** stops the output to the terminal but does not terminate the command. Entering **Control-Q** now resumes the output from the point at which it has been stopped. Entering **Control-C** now terminates the command.

Note that **Control-S** should be followed only with **Control-Q** or **Control-C**. No other console input is allowed following a **Control-S**.

2. In Stand-alone Mode:

When working in stand-alone mode, the LCD is refreshed when the display on row 4 is completed. The next display occurs on Row 1 of the LCD. In case of commands where large display is involved, such as 'Display Memory', the LCD displays a total of 16 bytes or 8 words at one time and waits for the user to enter <CR> to proceed with subsequent displays. Also there is an additional <CR> required in stand-alone mode for the monitor to return to Command entry prompt. However such commands can be terminated at any stage by entering <Esc>.

4.5 COMMANDS DESCRIPTION

This section describes the commands supported by ESA 86/88-3 monitor. The discussion holds good for either mode of trainer operation unless otherwise mentioned.



4.5.1 SUBSTITUTE MEMORY COMMANDS

FUNCTION

Substitute memory byte (S) and Substitute memory word (SW) commands are used to examine the contents of the selected memory locations. If the memory locations selected can be modified (i.e., user RAM locations), the contents can optionally be modified.

FORMAT

```
S[<address>]<CR> [,<new data>] * <CR>
SW[<address>]<CR> [,<new data>] * <CR>
```

OPERATION

1. Both commands operate in a similar fashion. To use either command, enter S or SW when prompted for command entry. Then enter the address of the memory location to be examined, followed by carriage return.
2. The monitor will now output the contents of the addressed location followed by a hyphen (the monitor's data entry prompt character) and a space to indicate that the addressed location can now be modified.
3. To modify the contents of this location, enter the new data value. Note that when using the SW command, the byte contents of the next consecutive memory location (addressed memory location +1) are output first, followed by the byte contents of the actual location addressed. Similarly, when updating memory contents using the SW command, the first byte entry will be written into the next consecutive memory location, and the second byte entry will be written into the addressed memory location.
4. After optionally modifying the contents of the addressed location, enter a comma to proceed with the command or <CR> to terminate the command. Entering a comma displays the byte or word data at the subsequent memory locations and prompts the user to optionally substitute the memory content with new data.

ERROR CONDITIONS

1. Trying to modify non-existent memory or ROM locations.
2. Entering non-Hex parameters as data.

EXAMPLES

1. Examine RAM location 2100H, relative to the DS register, modify the contents of location 2101H and 2102H and examine the contents of 2101H again.

```
.S DS:2100 <CR>
0000:2100  A5- ,
0000:2101  FF- B7 ,
0000:2102  FF- 5A ,
0000:2103  FF-<CR>
.
```



```
.S DS:2101 <CR>
0000:2101 B7- <CR>
.
```

Note: Here DS is set at 0000. Hence the segment value displayed will be 0000.

2. Examine ROM location FF00:9CH and trying to modify the same.

```
.S FF00:9C<CR>
FF00:009C FF- 44, ?<CR>
.
```

3. Examine word at location 2120 H relative to DS register. Assume that DS is set to 2000

```
.SW DS:2120 <CR>
2000:2120 A1F4 <CR>
.
```

Note: Here the location DS:2121H contains A1H while DS:2120 contains F4H

4.5.2 DISPLAY MEMORY COMMANDS

FUNCTION

Display memory byte (D) and Display memory word (DW) commands enable the user to output the contents of a block of memory to the LCD or to the console, either in byte or word format.

FORMAT

```
D <start address> [,<end address>]<CR>
DW <start address> [,<end address>]<CR>
```

OPERATION

Both the commands operate in a similar fashion. When operating in Serial Mode, these commands provide a line-formatted output of the memory block starting at the 'start address' and ending at the 'end address'. However in case of Stand-alone operation a total of 16 bytes or 8 words is displayed at one time and the monitor waits for a <CR> to refresh the output and display the next 16 bytes or 8 words of data if required.

Note: The 'end address' is always relative to the segment value specified with "start address", or implied with 'start address' (i.e. the current contents of CS register if no segment value are specified). Thus with each command execution, a maximum of 64K bytes or 32K words can be displayed.

To use either command, enter D (for byte output) or DW (for word output) when prompted for command entry. Then enter the start address of the memory block. Now, if only one byte or word is to be displayed, enter the carriage return. On the other hand, if a block of memory is to be displayed enter a comma and the end address of the range followed by <CR>.



The monitor will now output, beginning on the next line, the starting offset address, the data contents of that location and the contents of the consecutive memory locations up to the end address, if an end address is specified.

The line format is arranged such that any subsequent lines (if present) will begin with an offset address whose last nibble is zero. A line consists of a maximum number of 16 bytes or 8 words. In case of stand-alone operation the same amount of data as contained in one line is displayed on the LCD making use of all the 4 lines available.

In serial mode, the ASCII equivalents of the memory locations are also displayed at the end of a line of display. ASCII equivalents that cannot be displayed are represented by a ".".

The display memory commands can be cancelled, or the output can be stopped and resumed at any time by entering appropriate control parameters described in Section 4.4

ERROR CONDITIONS

1. Specifying an end address that is lesser than the offset value of the start address.

EXAMPLES

1. Display contents of location 140H relative to DS register. Assume DS is set to 0000.

```
.D DS:140 <CR>
0000:0140 A0
.
```

2. Display contents of locations 1000:10CH through 1000:125H.

```
.D 1000:10C,125 <CR>

          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

1000:010C                                F0 00 00 9C
1000:0110    EA BD 04 BA 00 F2 10 21 A4 35 14 17 FA F0 F1 F5
1000:0120    30 31 32 32 34 35
.
```

In addition to the above, the ASCII characters equivalent to the data bytes are also displayed on the right hand side of each line of output only in Serial Mode of operation. The non-existent ASCII equivalent data are represented by a '.'. In the stand-alone mode the output for the above command will be in the format described below.

```
.D1000:10C,125 <CR>

1000:010C
    F0 00 00 9C
```

The monitor now waits for the user to enter <CR> to refresh the display and output subsequent data bytes.



```

<CR>
1000:0110
    EA BD 04 BA 00 F2
    10 21 A4 35 14 17
    FA F0 F1 F5
<CR>
1000:0120
    30 31 32 32 34 35
<CR>
.

```

3. Display word at location 10C H relative to DS register. Assume DS is initialized to 2000.

```

.DW DS:10C<CR>
          0000 0002 0004 0006 0008 000A 000C 000E

2000:0100                                00F0
.

```

In case of stand-alone operation, the LCD will output

```

.DW DS:10C<CR>
2000:010C
    00F0
<CR>
.

```

For displaying a range of data words, the same procedure as with Display Byte is followed, with the only exception that **DW** replaces **D**. Note that ASCII equivalent characters are not displayed with word data.

4.5.3 EXAMINE/MODIFY REGISTER COMMAND

FUNCTION

The Examine/Modify Register (X) command is used to examine and optionally modify any of 8086/8088 CPU registers contents.

FORMAT

X [**<reg>**<CR> [**<new data>**/,]] <CR>

OPERATION

1. To use the Examine/Modify Register command, enter X when prompted for command entry.
2. To examine the current contents of all the registers, enter <CR>. Now the monitor will output the contents of all 14 registers.



3. If the contents of a particular register are to be examined or modified, enter the abbreviated register name after entering X and press <CR>. The abbreviations for the register names are shown in the table 4.2

Table 4.2 Register Abbreviation

Register Name	Abbreviation
Accumulator	AX
Base	BX
Count	CX
Data	DX
Stack Pointer	SP
Base Pointer	BP
Source Index	SI
Destination Index	DI
Code Segment	CS
Data Segment	DS
Stack Segment	SS
Extra Segment	ES
Instruction Pointer	IP
Program Counter*	PC

Now the monitor will output an equals sign '=', the current contents of the specified register, the data prompt character '-' and a space. If you wish to change the contents of this register, enter the new value followed by a comma, or a carriage return.

Entering a comma displays the contents of the next "sequential" register and opens it for optional modification. The register sequence is in the order as shown in the above table. This sequence is not circular i.e. if a comma is entered after the contents of the "last modifiable" register (viz. Flags register FL) is examined /modified, the command is terminated and the monitor returns to the command entry mode.

Entering a carriage return terminates the command.

*Program Counter cannot be modified using this command. To modify PC, the user will have to modify CS and IP registers, such that the PC will point to the desired effective address.

EXAMPLES

1. Examine the contents of all the registers.

```
.X <CR>
AX=1182 BX=A113 CX=000F DX=1242 SP=0100 BP=4020 SI=6020 DI=6F20
DS=0000 SS=0000 ES=0000 CS=0000 IP=0200 FL=F046 PC=00000
.
```

In case of stand-alone mode operation, the display format will be as follows.

```
.X <CR>
```



```
AX=1182  BX=A113
CX=000F  DX=1242
SP=0100  BP=4020
SI=6020  DI=6F20
```

The monitor now waits for a user strobe viz. <CR> to refresh the display and output subsequent data

```
<CR>
DS=0000  SS=0000
ES=0000  CS=0000
IP=0200  FL=F046
PC=00000
<CR>
.
```

2. Examine and modify the SP register and examine the next register i.e. BP

```
.XSP=0100 - 0110,
BP=4020-<CR>
.
```

4.5.4 MOVE MEMORY COMMAND

FUNCTION

Move Memory command (M) is used to copy a block of data from one area of the memory to another area within the CPU's memory space.

FORMAT

M <start address>,<end address>,<destination address> <CR>

OPERATION

1. To use this command, enter M when prompted for command entry. Now enter the three required parameters separated by commas.

The three parameters are the "start" and "end" addresses of the memory block to be copied and the start address of the destination block. In this command, the end address is relative to the segment value specified with 'start address' or implied with the 'start address' (i.e. the current contents of CS register if no segment value is specified). Consequently, no segment value is permitted with the 'end address' and with each command execution, a maximum of 64K bytes can be moved. The segment for the destination address is relative to the current CS value unless otherwise specified. Then enter <CR> to execute the command.

2. Now the monitor copies the contents of the memory block from "start address" to "end address", of consecutive memory locations beginning at 'destination address'. After moving the block of memory, monitor returns to the command entry mode.



Notes: Since Move operation is performed one byte at a time, M command can be used to fill a block of memory with a predefined constant. To do this, move the constant into the "start address" location using S command. Then M command is used with a "destination address" which is one greater than the "start address". Then the memory block from start address to "end address" +1 is filled with the value contained in "start address".

ERROR CONDITIONS

1. Trying to move data into a non-existent memory or ROM locations.
2. Specifying an 'end address' value that is less than the offset value of the "start address".

EXAMPLES

1. Move the contents of the locations 2000H through 2020H relative to CS register, to the memory block starting at 3000H relative to the DS register.

```
.M2000,2020,DS:3000<CR>
```

2. Fill memory locations 1000:2000H through 1000:2100H with constant AAH.

```
.S 1000:2000, <CR>  
1000:2000 BA- AA<CR>  
.M1000:2000,20FF,1000:2001<CR>
```

The validity of the Move Memory commands can be verified anytime using the Display Memory or Compare Memory commands.

4.5.5 FILL BYTE AND FILL WORD COMMANDS

FUNCTION

This command is used to fill a block of memory with constant byte or word data.

FORMAT

```
F<start address>,<end address>,<byte value><CR>  
FW<start address>,<end address>,<word value><CR>
```

OPERATION

1. Both commands operate in a similar way. Enter F or FW to fill byte or word data respectively when prompted for command entry. Now enter the starting address of the block and the ending address of the block separated by a comma.
2. Now enter a comma and the byte or word data to be filled. Enter <CR> to execute the command after which the monitor returns to the command entry mode.
3. The user may check the validity of this command by using Display Memory commands described above.



NOTE: The command will accept the last two Hex Characters entered as valid data in case of Byte operation and the last four Hex characters as valid data during Word operation. In case, less than four characters are entered during Word operation the command assumes the upper nibbles not entered to be 0.

ERROR CONDITIONS

1. Entering the ending address offset lesser than the starting address offset
2. Trying to enter a non-Hex value for the address or data.
3. Trying to fill non-existent memory or ROM locations.

EXAMPLE

1. Filling the memory locations from 0:2000 to 0:2050 with constant data byte 55
F0:2000,2050,55<CR>
2. Filling the memory locations from 0:5000 to 0:6000 with constant word data 1234
FW0:5000,6000,1234<CR>
3. Filling the memory locations from 0:5000 to 0:6000 with constant word data 00AA
FW0:5000,6000,AA<CR>

4.5.6 INPUT BYTE AND INPUT WORD COMMANDS

FUNCTION

The Input Byte (I) and Input Word (IW) commands are used to input (accept) a byte or word from an input port, and to display the accepted byte or word.

FORMAT

I <port address> <CR>[,] * <CR>
IW <port address> <CR>[,] * <CR>

OPERATION

1. Both the commands operate in a similar way. Enter I (to input byte) or IW (to input word) when prompted for command entry. Now enter the address of the port to be read followed by <CR>. Since the I/O space is only 64K bytes, no segment value is permitted with 'port address'.
2. The monitor now reads the byte or word at the specified port and displays it on the console or LCD. To proceed with the command, enter comma. Each subsequent comma increments the port address and displays the current data at the addressed port on a new line. Entering a carriage return terminates the command and the monitor returns to the command entry mode.



EXAMPLES

1. Input a byte from parallel I/O port at address FFE1H

```
.IFFE1<CR>
FFE1 - FA<CR>
.
```

2. Input a series of words from parallel I/O ports located at addresses FFE0H and FFE6H.

```
.IWFFE0<CR>
FFE0 - A2A2,
FFE2 - B2B2,
FFE4 - C2C2,
FFE6 - D2D2<CR>
.
```

Note that when using word input i.e. IW command, lower order addresses i.e. addresses on word boundaries are entered as port addresses.

4.5.7 OUTPUT BYTE AND OUTPUT WORD COMMANDS

FUNCTION

The Output Byte (O) and Output Word (OW) commands are used to output a byte or word to an output port.

FORMAT

```
O<port address> <CR> [<data>/,] * <CR>
OW<port address> <CR> [<data>/,] * <CR>
```

OPERATION

1. Both the output commands (O and OW) operate similarly. Enter O to output a byte, or OW to output a word when prompted for a command entry. Now enter the address of the port to which data is to be output followed by a <CR>. No segment value is permitted with the port address. (I/O address space is limited to 64K bytes).
2. After this, enter the data to be output in Byte or Word format as applicable or a comma if you wish to skip outputting data at this port. Entering a comma will increment the port address after which the monitor displays this address on a new line and the process continues.
3. To send data to the specified port, enter the byte or word data, followed by a comma or <CR>. Entering <CR> terminates the command after the command execution. Entering a comma sends the data to the specified port but does not terminate the command, and it permits subsequent data output to IO ports in succession. The users always have an option to either output data at a port, or skip the port and proceed with the subsequent I/O port. At any stage, entering <CR> terminates the command and returns to the monitor command prompt.



EXAMPLES

1. Program the parallel I/O port FFE0H for output by sending appropriate command byte to the status port FFE6H.

```
.OFFE6 <CR>
FFE6 - 80<CR>
.
```

2. Output a series of words to ports ranging from FF00 to FF06

```
.OFF00<CR>
FF00 - 1234,
FF02 - ABCD,
FF04 - 5555,
FF06 - 3676<CR>
.
```

4.5.8 COMPARE MEMORY COMMAND

FUNCTION

This command compares a block of memory with another block and displays differences in the locations when found.

FORMAT

C<start address1>,<end address1>,<start address2><CR>

OPERATION

1. To use this command enter C when prompted for command entry
2. Now enter the starting address of the first block of memory followed by a comma and the ending address of the block. Follow this with a comma and the starting address of the second block.

NOTE: In this command, **end address1** is always relative to the segment value specified with **start address1** or implied with **start address1** (i.e. the contents of CS register if no segment value is specified). Consequently, no segment value is permitted with the **end address1** and with each command execution, a maximum of 64K bytes can be compared. The segment for **start address2** is relative to the current CS value unless otherwise specified.

3. Pressing <CR> starts execution of the command. The monitor will compare subsequent data at each corresponding location of the blocks and display differences when found in the following pattern. The display format for stand-alone mode of operation is described below

Segment1:offset1 - data Segment2:offset2 - data

4. The monitor returns to command entry mode at the end of command execution



ERROR CONDITIONS

1. Entering the ending address offset lesser than the starting address offset
2. Trying to enter a non-Hex value for the address or data.
3. Entering a value for the offset of the second address such that the range of locations left in the segment from this offset does not match the range to be compared.

EXAMPLES

1. Compare a block of memory beginning at 0:2000 to 0:2050 with a block beginning at 0:3000

```
.C0:2000,2050,0:3000<CR>
```

.

This response shows that there was no mismatch.

2. Compare a block of memory beginning at 0:2500 to 0:2550 with a block beginning at 0:3000

```
.C0:2500,2550,0:3000<CR>
```

```
0000:2505 - 45          0000:3005 - 76
0000:2515 - 21          0000:3015 - 16
0000:2532 - 45          0000:3032 - A3
0000:2544 - 1A          0000:3044 - 22
```

.

This response shows that there was a mismatch at 4 locations.

In case of stand-alone operation, the command entry format is similar to that in the serial mode. The output in case of example 2 will be as follows.

```
0000:2505 - 45
0000:3005 - 76<CR>
0000:2515 - 21
0000:3015 - 16<CR>
```

The display will now be refreshed and the monitor will output the following result.

```
0000:2532 - 45
0000:3032 - A3<CR>
0000:2544 - 1A
0000:3044 - 22<CR>
```

.

4.5.9 GO (EXECUTION) COMMAND

FUNCTION

Go (G) command is used to transfer control of the CPU from the Monitor to a user's program.



FORMAT

G <CR> [**<start address>** [,**<breakpoint address>**]]<CR>

OPERATION

1. To use this command, enter G when prompted for command entry followed by a <CR>. The monitor now outputs the current CS and IP register contents in the Segment:Offset format.
2. If the user wishes to start program execution from an alternate starting address, the user has to enter the same. Segment specification is not necessary if the segment value of the program start address is the same as the current code segment.
3. The user may use the Breakpoint facility by entering a comma followed by a break point address at which program execution needs to be stopped. No segment value is permitted with breakpoint address. Thus the default segment value for the break-point address is either the segment specified with the "start address" or the current CS register content if a segment value is not specified. Breakpoint implementation is discussed below in detail.
4. To begin program execution, enter carriage return. CPU Control is now transferred from the monitor to the user program. Program termination is entirely dependent on the instructions entered by the user.
5. To exit from the executing program and to return control to the monitor the user has the following options.
 - ♦ At the logical termination of the user's program the user should enter the instruction **INT 03** (Hex Code = CC). This instruction will perform the Type 3 interrupt service routine embedded in the ESA 86/88-3 Monitor, which will break the program at that address, display all the current register contents and return to command entry prompt.
 - ♦ The user can use the Breakpoint facility provided with the GO command and can break the program at any desired address. With this option, the user program breaks at the specified address and control is transferred back to the monitor. Breakpoint facility is discussed in detail below.
 - ♦ If the user program ends with a **HLT** instruction (Hex Code = F4) or continues to execute in an endless loop, the user may have to RESET the trainer. However upon RESET, any previous information of the program contained in segment and status registers is lost and they are reinitialized to their default values.

4.5.10 BREAKPOINT FACILITY

GO command permits the optional specification of a "breakpoint address". When program reaches the breakpoint address, control is transferred to the monitor and contents of all registers are saved. Before returning to command entry prompt, the monitor outputs the message BR@[†] followed by a display of all processor registers, the breakpoint address in Segment:Offset format, the machine code and disassembled line of the program instruction at that address.



If a subsequent GO command is entered, the current address displayed will be the break-pointed address. The user can resume execution of the program from this address by just entering <CR>.

While implementing breakpoint facility the monitor saves the instruction at the breakpoint address, and replaces it with an interrupt before transferring control to the user's program. When the program reaches the breakpoint address, control is returned to the monitor. The monitor saves all CPU registers, restores the break-pointed instruction and outputs the previously described pattern. It then issues the command entry prompt allowing the user to modify any of the registers, memory locations before resuming the execution of the program.

From the above discussion, it is clear that

- ♦ The user can specify a Breakpoint address only when a comma precedes it.
- ♦ The segment value for the breakpoint address is always relative to the “start address” segment value.
- ♦ An instruction in the read-only memory cannot be break-pointed.
- ♦ Break point address must be specified each time when a program to be break-pointed is executed.

[†]BR@ is displayed on the console only i.e. it appears only in the serial mode. In stand-alone mode the monitor proceeds with the registers display directly.

EXAMPLES

1. Transfer control to the program at 2000H relative to CS register

```
.G<CR>
.G 0000:0000 2000<CR>
.
```

2. Transfer CPU control to a program starting at location 20:300H and breakpoint the instruction at location 20:3F2H.

```
.G<CR>
.G 0000:0000 20:300,3F2<CR>
BR@
```

```
AX=1182 BX=A113 CX=000F DX=1242 SP=0100 BP=4020 SI=6020 DI=6F20
DS=0000 SS=0000 ES=0000 CS=0020 IP=03F2 FL=F002 PC=005F2
0020:03F2 55                      PUSH BP
.
```

In Stand-alone mode of operation, the output format on the LCD with respect to the above command will be as described below.

```
.G<CR>
.G 0000:0000
20:300,3F2<CR>
```



The monitor refreshes the LCD and displays all the CPU registers in the following format. In this mode, the monitor waits for user strobes wherever necessary.

```
AX=1182 BX=A113
CX=000F DX=1242
SP=0100 BP=4020
SI=6020 DI=6F20
<CR>
```

```
DS=0000 SS=0000
ES=0000 CS=0020
IP=03F2 FL=F002
PC=005F2
<CR>
```

```
0020:3F2
PUSH BP
55
<CR>
```

.

3. Attempting to breakpoint an instruction in the EPROM area. As shown below, this will result in an error.

```
.G<CR>
.G 0000:0000 F000:100,122<CR>
?G 0000:0000 20:300,3F2
```

NOTE: All underlined lines in these examples are either overwritten by the statements following or are inserted in the text merely for illustration.

4.5.11 SINGLE STEP COMMAND

FUNCTION

Single step (N) command is used to execute single instruction of a program. After the execution of each instruction executed, CPU control is returned to the monitor from the program being executed.

FORMAT

N<CR> [<start address>],[/<new address>]* <CR>

OPERATION

1. To use the single step command, enter N with <CR> when prompted for command entry. Now the monitor will output the current contents of CS & IP registers in Segment:Offset format.



2. If the program is to be executed from an address other than that displayed, the user must enter a new 'start address' followed by a comma. The segment value of the 'start address' is implied to be the same as the current value of the CS register unless it is explicitly entered.
3. The monitor now outputs the current content of the CPU registers followed by a line of disassembled code at the start address. To proceed with execution of this instruction, enter another comma.
4. The addressed instruction is now executed and CPU control returns to the monitor. Before this the monitor saves all the registers, displays them and outputs a line of disassembled code at the address of the next subsequent instruction. Now the user can enter a comma and proceed with executing the instruction at the displayed address or enter a new address at which an instruction is to be single stepped. Note that the monitor does not output a command entry prompt, although at this stage control rests with the monitor. Hence entry of any other command at this stage will result in an error.
5. Each time a comma is entered, the addressed instruction is executed and the address and instruction byte contents of the next instruction to be executed along with all registers and a disassembled line of next instruction code is displayed. A new start address can be specified after executing any instruction.
6. A carriage return, after executing an instruction terminates the command and returns the monitor to command entry mode. Modification of any memory or register contents can be done while single stepping through a program only by using a <CR>. However since the current command is terminated, the user will have to re-enter the single-step command and proceed.

RESTRICTIONS

1. An instruction that is part of a sequence of instructions that switches between stack segments (i.e. changes the SS and SP register contents) can not be single stepped.
2. If an interrupt occurs prior to the completion of a single-stepped instruction or if a single-stepped instruction generates an interrupt, then CS and IP registers will contain the address of the interrupt service routine when the monitor regains control. Consequently, a Type 3 (Breakpoint) interrupt instruction (CCH) should not be single-stepped (since its execution will cause the program execution sequence to enter the Monitor).

EXAMPLES

1. Single step the first three instructions of a program starting from location 0:2000H onwards and then branch off the execution sequence to location 100:3000H and execute the single instruction at that location.

.N<CR>

.N 0000:0000 0:2000,

AX=0000 BX=0000 CX=0000 DX=0000 SP=0100 BP=0000 SI=0000 DI=0000
 DS=0000 SS=0000 ES=0000 CS=0000 IP=2000 FL=0000 PC=02000
 0000:2000 BA E6 FF MOV DX,FFE6

/
 BR@



```
AX=0000 BX=0000 CX=0000 DX=FFE6 SP=0100 BP=0000 SI=0000 DI=0000
DS=0000 SS=0000 ES=0000 CS=0000 IP=2003 FL=F002 PC=02003
0000:2003 B0 80          MOV    AL,80
```

/
BR@

```
AX=0080 BX=0000 CX=0000 DX=FFE6 SP=0100 BP=0000 SI=0000 DI=0000
DS=0000 SS=0000 ES=0000 CS=0000 IP=2005 FL=F002 PC=02005
0000:2005 EE          OUT    DX,AL
```

/
BR@

```
AX=0080 BX=0000 CX=0000 DX=FFE6 SP=0100 BP=0000 SI=0000 DI=0000
DS=0000 SS=0000 ES=0000 CS=0000 IP=2006 FL=F002 PC=02006
0000:2006 FF          DB      FF
100:3000,
BR@
```

```
AX=0080 BX=0000 CX=0000 DX=FFE6 SP=00FE BP=0000 SI=0000 DI=0000
DS=0000 SS=0000 ES=0000 CS=0100 IP=3001 FL=F002 PC=04002
0100:3001 50          PUSH   AX
<CR>
```

.

For better understanding and familiarization of this command, single step the example programs given in Chapter 10 following the above discussion and observe the results.

Note: The above description of this command is applicable to serial mode operation where the command output can be seen on the console. The command will follow a similar execution sequence in stand-alone mode operation also, except for a difference in the register display format. An extra <CR> will be required to proceed with the command since the LCD output will be refreshed after a strobe from the user.

4.5.12 HELP COMMAND (H)

FUNCTION

This command lists ESA 86/88-3 Monitor commands with their valid syntax in Stand-alone mode of trainer operation.

FORMAT & OPERATION

H[<Command mnemonic>]<CR>

1. When prompted for command entry entering H with <CR> lists ESA 86/88-3 Monitor commands with a description of their valid syntax on the LCD. Thus the user can use this command as a ready reference source on the monitor commands.
2. If the command syntax of a particular command is required, then the user should enter H followed by the command Mnemonic.



For e.g. entering HS with <CR> will output the syntax of the Substitute memory command.

After displaying the command, the monitor waits for a user strobe viz. <CR> to display the next command and its syntax. Pressing <Esc> will terminate the command and the monitor returns to command entry mode.

The command syntax appearing with the description of this command is the same as that of the Command Help appearing in Serial Mode of operation.

4.6 MORE MONITOR COMMANDS

1. Refer Chapter 7 of this manual for a detailed description of ESA 86/88-3 EPROM Programmer system invoked by the following command.

P (Invoke Programmer) **P<CR>**

The EPROM Programmer software further supports commands to Read, Blank Check, Verify and/or Program the popular EPROMS 2716 through 27512 and the on-chip code memory of 8x51 Microcontrollers.

2. Refer Chapter 8 of this manual for a detailed description of ESA 86/88-3 Symbolic One-line Assembler and a discussion of the following Monitor commands in support of this assembler.

A (Invoke Assembler) **A[address]<CR>**

LL (List Labels) **LL<CR>**

LC (Clear Labels) **LC<CR>**

Z(Disassembly) **Z[<start address>[,<end address>]]<CR>**



CHAPTER 5

SYSTEM DESIGN DETAILS **AND MEMORY ADDRESSING**

This chapter gives an insight into the hardware design details of ESA 86/88-3 Microprocessor trainer. The onboard interfacing options and external interfaces supported by the trainer are discussed in subsequent chapters. Appendix A gives the complete schematics, Appendix B gives the connector details and Appendix C has the component layout diagram.

The design details discussed in this chapter include:

- a) CPU
- b) System timing and Wait State Logic
- c) Memory addressing
- d) Interrupt system
- e) 8087 Co-processor (NDP)
- f) Bus expansion

5.1 CPU

ESA 86/88-3 can be operated with either the 8086 CPU or 8088 CPU. A single DIP Switch setting as explained in Section 2.1.1 makes the processor selection. In either case, the CPU is configured to operate in Maximum mode.

5.1.1 CPU RESET

An on-board RESET key provides the reset signal to the CPU through the 8284A-clock generator. The RESET key gives an RSTIN* signal to the 8284 and its RESET output is used to reset the CPU and the rest of the system synchronously. This signal is available on the bus connector also and can be used to RESET off-board peripherals.

5.1.2 CPU CLOCK

A 24 MHz Oscillator is the clock source for 8284A-clock generator. The 8284A divides this frequency by three and produces an 8 MHz clock with 33 % duty cycle as required by 8086/8088. Further, 8284A provides a 4 MHz PCLK with 50% duty cycle, which can be used as a clock source for onboard peripherals. Both these clock outputs are available on the bus connector and may be used as a source clock frequency to external peripherals.

The user can also opt for a 15 MHz crystal as the clock source for the Clock generator and operate the CPU at 5 MHz. This option may be utilized by placing a jumper at JP-23, which selects the 15MHz crystal as input to the 8284-clock generator.

NOTE: Placing a jumper at JP-22 corrects the clock output of the 8284 to TTL compatible levels.



5.1.3 CPU ADDRESS BUS

Latches (74LS373s) at U18, U19 and U30 are used to latch the address with the help of ALE signal. These latches are always enabled since there is no provision for external bus master. Also note that BHE*, Status signals - S0*, S1* and S2* are also latched.

As the CPU operates in maximum mode, the 8288-bus controller is used to decode the status signals and provide all the control signals.

5.1.4 CORRECTED BHE*

8086 CPU generates a BHE* signal on pin 34 and this signal is used to access the devices residing on the upper data bus. However, 8088 CPU always drives a high signal on pin 34 as its access is always over the lower data bus. To accommodate this difference, a corrected BHE* signal is used in the rest of the system. This corrected BHE* signal is either the latched BHE* signal (for 8086) or the inverted latched A0 signal (for 8088).

This selection is accomplished by tri-state buffer 74LS125 at U50, the enable signals for which are derived from the setting of SW7 of the DIP Switch. This switch setting indicates the selection of 8086/8088 CPU. Thus if 8086 is selected the LS125 will inhibit inverted A0 and pass latched BHE* signal as corrected BHE* signal. On the other hand, if 8088 is selected, the 74LS125 will inhibit latched BHE* (which will be always high) and pass the inverted A0 signal as the corrected BHE* signal.

5.1.5 CPU DATA BUS

Bi-directional buffers (74LS245s) at U16, U17 and U21 are used to buffer the CPU data bus. Note that one buffer (74LS245 at U17) is used as swap buffer to route the upper data bus to the lower data bus when 8088 is selected. The enable signals for the bus are derived from corrected BHE*, A0, DEN* (from 8288 bus controller) and 86SEL* and 88SEL signals.

This logic implemented by 74LS32 at U43 is as follows:

CPU	DEN*	A0	CORRECTED BHE*	ENABLE FOR		
				Upper Data Bus Buffer	Lower Data Bus Buffer	Swap Buffer
8086	HIGH	X	X	NO	NO	NO
	LOW	LOW	LOW	YES	YES	NO
	LOW	LOW	HIGH	NO	YES	NO
	LOW	HIGH	LOW	YES	NO	NO
8088	HIGH	X	X	NO	NO	NO
	LOW	LOW	HIGH	NO	YES	NO
	LOW	HIGH	LOW	NO	NO	YES

(X: Don't Care; The combinations not shown above never occur)



5.2 SYSTEM TIMING AND WAIT STATE LOGIC

The time-base for CPU operation is derived from an 8284A Clock generator. The CPU may be operated at a frequency of 8 MHz or 5 MHz, which is the clock output from the Clock generator with 33% duty cycle. The Clock generator also generates a clock output PCLK with 50% duty cycle. This clock frequency is 4 MHz when the input to 8284 is from 24 MHz Oscillator and 2.5 MHz when the input to 8284 is from 15MHz crystal.

The Clock (CLK*) signal is utilized by the 8042 Universal Peripheral Interface as its clock input. The serial interface controller SCN2681 Dual UART uses a separate crystal of 3.6864 MHz as its clock frequency source. The 8284 Clock generator also generates the READY and RESET signals to the CPU synchronously. Ready timing and Wait States implementation is described below.

All instruction cycles in ESA 86/88-3 trainer are run with one Wait State regardless of whether they are memory or I/O cycles. The Wait State logic is implemented by a 74LS76 at U46. The ALE* signal is connected to the CLEAR input of the flip-flop. Hence in every instruction cycle the Q* output of 74LS76 goes low on the rising edge of T2 State de-asserting the RDY1 signal to 8284A. On the next low-to-high transition of the clock, the Q* output toggles again reasserting the RDY1 signal to 8284A. Thus one Wait State is introduced during the execution of every instruction cycle.

The PRESET input to 74LS76 is pulled up to VCC and can be controlled by off-board devices (memory or I/O) to introduce as many wait states as required. This line is designated as EXTRDY and is available on the system connector J1. When this line is pulled low, Ready input to 8284A is de-asserted and when this line is high or open there is no influence on the Ready line. This scheme permits off-board devices to introduce Wait States if required or let the system operate with one Wait State for all instruction cycles.

5.3 MEMORY ADDRESSING

ESA 86/88-3 Memory is arranged in Odd and Even banks of memory so that 16-bit Read or Write operations can be achieved within a single cycle. The lower 8 bits of the data bus are connected to the even memory bank while the upper 8 bits are connected to the Odd memory bank.

ESA 86/88-3 has six 32-pin JEDEC compatible slots labeled as U6-U11 for memory ICs. Sockets U6 & U9 are populated with EPROMs 27C010 (128K x 2 = 256K) containing System firmware. Sockets U7 & U10 are populated with static RAMs 628128 (128K x 2 = 256K). This RAM can be upgraded by populating RAM ICs 628512 (512K) in these slots and by making suitable jumper settings. Sockets U8 & U11 are provided for memory expansion using either RAM or ROM.

Memory configuration and jumper settings for various memory ICs are described in Chapter 2. RAM area starts from 0:0000H. However the RAM area from 0:0000H to 0:1FFFFH is used by the system for Interrupt vectors, Stack and Assembler data tables. Thus user RAM area starts from location 0:2000H onwards.



5.3.1 MEMORY MAP

With the versatile memory options offered in ESA 86/88-3 Trainer, the user can collectively configure the memory devices to realize the full 1MB addressing capability of 8086/8088 CPU.

PROM Area: (Sockets U6 & U9)

PROMS Selected	Address Range (in Hex)
27C512	F0000 - FFFFF
27C010/27C020/27C040	E0000 - FFFFF

RAM area: (Sockets U7 & U10)

Device	Jumper Position @ JP-20	Address range (in Hex)
628128/628512	A-B	00000 - 3FFFF
628512	B-C	00000 - 7FFFF

Expansion Memory area: (Sockets U8 & U11)

Device	Jumper Position @ JP-20	Address range (in Hex)
27C512	A-B	40000 - 5FFFF
	B-C	80000 - 9FFFF
27C010/628128	A-B	40000 - 7FFFF
	B-C	80000 - BFFFF
27C020/628512	A-B	40000 - BFFFF
	B-C	80000 - DFFFF

Optional battery backup provision is available for RAM and Expansion Memory using an onboard 3.6V Ni-Cd. cell. Refer Appendix C for Component layout diagram.

5.4 INTERRUPT SYSTEM

Hardware Interrupts: The 8086/8088 CPU supports two hardware interrupts, NMI and INT.

The NMI or Type 2 Interrupt (Non-Maskable Interrupt) is connected to the BREAK key on the trainer. Providing the vectoring information for this interrupt is entirely left to the user. To make use of this interrupt, the user should enter the Instruction Pointer Word of the Interrupt service routine at address 0:0008H and the Code Segment Word at address 0:000AH. Then pressing the BREAK key on the trainer will cause an NMI to occur and the CPU control is transferred to the NMI service routine specified by the address entered by the user.

An example illustrating the use of BREAK key is given in Chapter 10.

The INT (maskable interrupt) is connected to the INTR output of 8259A. Ref section 2.1.7 for details of interrupt selections. The system expansion connectors also provide the facility to cascade external 8259A PICs with the onboard interrupt controller.



Internal Interrupts: INT 3 can be used by user programs to return control to the monitor. INT 1 is reserved by the monitor and is used for Single Stepping. Other internal interrupts are available to user.

5.5 8087 NUMERIC DATA PROCESSOR

On-board provision is made for 8087 Co-processor. The CPU can be either 8086 or 8088. To install 8087, switch off the power and populate 8087 in the socket U32. No other hardware changes are necessary. (Refer the component layout diagram in Appendix C).

The NDPINT output from the NDP can also be used as an on-board interrupt source to the interrupt controller (Refer Section 2.1.6)

5.6 BUS EXPANSION

The buffered address, data and control signals are provided on the bus connectors J1 and J2. The Bus connector details are given in the tables below. The user may use this information and implement bus interfacing accordance to individual requirements. The signal definitions and details of all connectors and headers are given in Appendix B.

J1-CONNECTED TO SYSTEM BUS

PIN NO	SIGNAL	PIN NO	SIGNAL
1	GND	2	GND
3	Vcc	4	Vcc
5	Vcc	6	Vcc
7	GND	8	GND
9	GATE1	10	GATE2
11	CONCLK	12	RESET
13	Timer0	14	PCLK
15	CINT0*	16	CINT1*
17	CINT2*	18	CINT3*
19	CINT4*	20	CINT5*
21	CINT6*	22	CINT7*
23	LS0	24	INTA*
25	LS1	26	MWTC*
27	LS2	28	IOWC*
29	Gate0	30	CAS0
31	CLK0	32	CAS1
33	LOCK*	34	CAS2
35	MRDC*	36	AMWTC*
37	IORC*	38	AIOWC*
39	ALE	40	EXTRDY
41	CLK1	42	CLK2
43	GND	44	GND
45	Vcc	46	Vcc
47	TIMER1	48	TIMER2
49	GND	50	GND

J2-CONNECTED TO SYSTEM BUS

PIN NO	SIGNAL	PIN NO	SIGNAL
1	GND	2	GND
3	BA0	4	CBHE *
5	BA1	6	BA2
7	BA3	8	BA4
9	BA5	10	BA6
11	BA7	12	BA8
13	GND	14	BA9
15	BA10	16	BA11
17	BA12	18	BA13
19	BA14	20	BA15
21	BA16	22	GND
23	BA17	24	BA18
25	BA19	26	NC
27	NC	28	CRTC
29	NC	30	GND
31	BD0	32	BD1
33	BD2	34	BD3
35	BD4	36	BD5
37	BD6	38	BD7
39	GND	40	GND
41	BD8	42	BD9
43	BD10	44	BD11
45	BD12	46	BD13
47	BD14	48	BD15
49	GND	50	GND



CHAPTER 6

ONBOARD PERIPHERALS AND INTERFACING OPTIONS

ESA 86/88-3 Microprocessor trainer features several onboard peripherals for a wide range of user applications, some of which are standard features of all ESA Trainers. Additionally, ESA 86/88-3 has provision for some more interfacing options such as DAC, ADC, Parallel Printer, etc.

This chapter introduces the user to the following onboard peripherals and interfaces and their applications along with the I/O address map, peripheral details and a summary of ESA86/88-3 connectors.

- ♦ 8255A Programmable Peripheral Interface
- ♦ 8253 Programmable Interval Timer
- ♦ SCN 2681 Dual UART
- ♦ 8259 Programmable Interrupt Controller
- ♦ 8042 Universal Peripheral Interface
- ♦ LCD Interface
- ♦ 8-Channel 12-Bit A/D Converter
- ♦ Onboard D/A converter
- ♦ Parallel Printer Interface

This trainer also supports ESA EPROM Programmer interface; this is discussed in detail in Chapter 7

6.1 I/O ADDRESSING AND I/O MAP

I/O decoding is implemented using the decoding ICs at U22 & U23. The following table summarizes the I/O mapped I/O address assignments and usage of all the peripherals of ESA 86/88-3 and also their function.

Note that some of the peripherals are fully available to the user while others are used to control interfaces such as A/D converter, DAC, parallel printer interface and LCD interface. However, the port lines from these peripherals are available on bus connectors and the user may utilize them suitably. (Refer Component Layout Diagram in Appendix C to locate these peripherals and controllers).

NOTE

1. The addressing specifications for SCN 2681 Dual UART are specified later in this chapter in Section 6.4. .
2. The DIP Switch used for system configuration is also mapped within the I/O Space of the CPU (shown in the following table)



TABLE 6.1 ESA 86/88-3 I/O ADDRESS MAP

PERIPHERAL/ CONTROLLER	PORT ADDRESS	DEVICE REGISTER	USAGE
DIP Switch	FFED	----	System configuration
8255-PPI High at U42 Mapped on Even Bus	FFE7	Control Port	24 I/O lines available to user for interfacing.
	FFE1	High Port A	
	FFE3	High Port B	
	FFE5	High Port C	
8255-PPI Low at U31 Mapped on Odd bus	FFE6	Control Port	24 I/O lines available to user for interfacing.
	FFE0	Low Port A	
	FFE2	Low Port B	
	FFE4	Low Port C	
8253-Programmable Interval Timer at U5	FFFF	Command Port	All timers are available to the user. The signals are brought out to Connector J1 and also to a header
	FFF9	TIMER 0	
	FFFB	TIMER 1	
	FFFD	TIMER 2	
8259A-Programmable Interrupt Controller at U4	FFF4	Command/Status Reg.	Provides interrupt vectors for 8 Onboard / External interrupt sources
	FFF6	Data port	
8255-3 at U1	FFD7	Control Port	I/O lines used for interfacing Centronics Parallel printer.
	FFD1	Port A	
	FFD3	Port B	
	FFD5	Port C	
8255-4 at U26	FFD6	Control Port	I/O lines used for interfacing D/A and A/D converters
	FFD0	Port A	
	FFD2	Port B	
	FFD4	Port C	
8255-5 at U27	FFDE	Control Port	I/O lines used for interfacing LCD.
	FFD8	Port A	
	FFDA	Port B	
	FFDC	Port C	
8042-Universal Peripheral Interface at U28	FFEB	Command Port	Interfacing with PC keyboard for stand-alone mode operation
	FFE9	Data port	

6.2 PROGRAMMABLE PERIPHERAL INTERFACES

ESA 86/88-3 features 5 onboard 8255A Programmable Peripheral Interface devices. Of these, two 8255s placed at U31 and U42 provide 48 I/O lines that are entirely available to the user. The peripheral at position U42, resides on the upper data bus, while the peripheral at U31 resides on the lower data bus. Each 8255A consists of three 8-bit I/O data ports (designated as Ports A, B and C) and one write-only Control Register. All the ports can be addressed individually and configured independently as either input or output.

The I/O address assignment can be found in the I/O map given in the section 6.1



The signals of the Port A, Port B and Port C of the 8255 at U31 are brought out to the FRC Connector J8 and the corresponding signals of the 8255 at U42 are brought out to FRC Connector J9. The pin assignments for the individual port bits are given in Appendix B.

NOTE: Port bits PC0, PC3 of U31 and PC3 of U42 can act as interrupt sources allowing the operation of 8255As in Mode 3 (Refer Section 2.1.7)

6.3 PROGRAMMABLE INTERVAL TIMER

ESA 86/88-3 has an on-board Programmable Interval Timer 8253 positioned at U5. 8253 has one command port and three data ports viz. Timer 0, Timer 1 and Timer 2 providing for three independent programmable timers. All of these are made available to the user. The outputs of the timers, along with their clock and gate inputs are available on the header J10. Refer to the connector details in Appendix B.

The I/O address assignments can be found in Section 6.1. The outputs from Timer 1 and Timer 2 can be used as on-board interrupt sources. Refer section 2.1.7.

6.4 SERIAL INTERFACE CONTROLLER

Serial communication in ESA 86/88-3 Microprocessor trainer is implemented using SCN 2681 Dual Universal Asynchronous Receiver Transmitter (DUART). The trainer is capable of communicating with serial terminals using either RS 232C or RS 485 standards.

SCN 2681 DUART provides two independent full Duplex Asynchronous Receivers/Transmitters in a single package. It interfaces directly with microprocessors and may be used in a polled or interrupt driven system. The operating mode and data format of each channel can be programmed independently, and each receiver and transmitter can select its operating speed as one of eighteen fixed baud rates, a 16 x clock derived from a programmable counter/timer, or an external 1 x or 16 x clock.

The clock input to the DUART is an onboard 3.6864 MHz crystal. Channel A of the DUART is programmed for serial communication using RS232C standard while Channel B is programmed for RS485 communication. The interfacing requirements and connector details for either operation are given in Appendix E. The Baud rate selection for serial communication is made by Dip Switch settings and up to 16 different standard baud rates can be established. Refer Chapter 2 for the look up table for baud rate selection.

Separate serial connections are provided on the trainer for interfacing with RS 232C and RS 485 ports.

J4: 9-pin Female type Connector for RS 232 Communication

J5: 9-pin Male type Connector for RS 485 Communication

To establish serial communication using ESA 86.88-3, the user has to interface the serial terminal with either of these connectors using appropriate serial cable

The following table gives the addressing information with respect to the various control registers of SCN 2681 DUART. Although the serial monitor initializes the device, the user may use the following information to re-configure the device to suit individual needs



SCN 2681 Register Description & Addressing

Address	Read Operation (RDN = 0)	Write Operation (WRN = 0)
FFA0	Mode Register A (MR1A, MR2A)	Mode Register A (MR1A, MR2A)
FFA2	Status Register A (SRA)	Clock select register A (CSRA)
FFA4	-- Reserved --	Command register A (CRA)
FFA6	Rx Holding Register A (RHRA)	Tx Holding register A (THRA)
FFA8	Input Port Change Register (IPCR)	Auxiliary Control Register (ACR)
FFAA	Interrupt Status Register (ISR)	Interrupt Mask register (IMR)
FFAC	Counter/Timer Upper (CTU)	C/T Upper register (CTUR)
FFAE	Counter/Timer Lower (CTL)	C/T Upper Lower register (CTUL)
FFB0	Mode Register B (MR1B, MR2B)	Mode Register B (MR1B, MR2B)
FFB2	Status Register B	Clock select register B (CSRB)
FFB4	-- Reserved --	Command Register B (CRB)
FFB6	Rx Holding Register B (RHRB)	Tx Holding Register B (THRB)
FFB8	-- Reserved --	-- Reserved --
FFBA	Input Port Register	Output Port Configuration register (OPCR)
FFBC	Start Counter Command	Set Output Port Bits Commands
FFBE	Stop Counter Command	Reset Output Port Bits Commands

6.5 PROGRAMMABLE INTERRUPT CONTROLLER

ESA 86/88-3 features an onboard Programmable Interrupt controller 8259A positioned at U4. The INTR output of this device is connected to the INT input of the CPU. Interrupt inputs to 8259A can be on-board sources or off-board sources. Refer Section 2.1.7 for details of interrupt sources selection.

The user can program the interrupt controller to generate either interrupts to the CPU from eight onboard or external sources along with the corresponding vectoring information for the interrupt service routines. Provision is also made on the Bus connectors for cascading an external 8259A with this interrupt controller.

An example program using 8259A to generate interrupt from onboard sources is given in Chapter 10. The I/O address assignments are given in Section 6.1.

NOTE: The system firmware does not initialize the 8259A to any mode. Thus user program must configure this device appropriately before using it.



6.6 8042 UNIVERSAL PERIPHERAL INTERFACE

ESA 86/88-3 Trainer is interfaced with a PC Keyboard and LCD for operation in stand-alone mode. The PC Keyboard is interfaced over a DIN connector and is controlled by an 8042 Universal Peripheral Interface. The UPI is a general-purpose controller that allows the user to develop customized solutions for peripheral device control.

The addressing information for 8042 UPI onboard ESA 86/88-3 is given in Section 6.1. The UPI uses buffered Clock (CLK*) from the Clock generator. Keyboard reading is implemented by polling the command/status port of 8042. User can read the keyboard in polling mode by checking the status of output buffer register. The keyboard sends scan codes for the respective keys pressed. The scan codes for the keys can be referred in the PC AT reference manual. The UPI is programmed for encoding either 101 or 84 keys PC keyboard.

6.7 LCD INTERFACE

In the stand-alone mode, an LCD is used as an output terminal for working with ESA 86/88-3 Trainer. The display is initialized as follows.

- 20 Digits, 4 Lines, Left entry display.

LCD module comprises two registers. Instruction and Data register. Three control signals RS, R/W, and E determine the operating status of LCD.

E = 1	:	For any operation with the LCD
RS = 1	:	Operation with Data Register
RS = 0	:	Operation with Instruction Register
RW = 0	:	Write to LCD
RW = 1	:	Read from LCD

The control and data lines to the LCD are controlled by the 8255 positioned at U27. These lines are brought to the flow-strip at J11 where the LCD is inserted. An example program to initialize the LCD is given in Chapter 10.

6.8 8-CHANNEL 12-BIT A/D CONVERTER

ESA 86/88-3 features an optional onboard 8 Channel, 12-bit ADC. The interfacing circuit consists of an analog Multiplexer that enables signal acquisition from up to eight different analog sources to be acquired. This ADC can accept either unipolar signals in the range 0 to 10V, or bipolar signals of -5V to +5V or in the range of -10V to +10V. The voltage span can be selected by placing suitable jumpers. By adding a single resistor in shunt with the input, current signals of 0 to 20mA or 4 to 20mA can also be directly handled by this interfacing circuit.

This circuit is built around the industry standard Fast ADC, AD1674 built in reference, clock and sample/hold circuit which completes a conversion in as little as 10μSec. This interfacing finds extensive use in the fields of analog measurements, transducer interfacing, Industrial monitoring and so on.

Circuit description

Please refer to the schematic diagram presented in Appendix A. For using the ADC, the user must connect +12V and -12V supplies to the trainer at the Power connector J3. The different modes of operation of the ADC and its configuration for different voltage spans are given below.



Single channel/multi-channel operation

Jumper JP19 decides whether the circuit is intended for single channel or 8-channel operation. For single operation, no Multiplexer is used and JP19 is left OPEN. Input signal is given at screw terminal TP. For multi-channel operation, jumper JP19 is closed. Signals to the ADC can now be given from up to eight different channels through the Multiplexer. The Multiplexer is used to make the channel selection by using Port lines PC4 to PC7 of 8255-4 at U26. Input signals are applied to any desired channel at screw terminals provided at J12. The channel selection is as shown in the table below.

P2C7	P2C6	P2C5	P2C4	Channel Selected
0	*	*	*	None
1	0	0	0	0
1	0	0	1	1
1	0	1	0	2
1	0	1	1	3
1	1	0	0	4
1	1	0	1	5
1	1	1	0	6
1	1	1	1	7

ADC interfacing to 8255

The converted data is latched into the latches 74LS374 (U40 and U41). P3A1 OF 8255 at U1 enables or disables these latches. When P3A1 is high the output of the latches are tri-stated. When P3A1 is low, data is available at port bits P4B0 through P4B7 and P4C0 through P4C3 of the 8255 at U26. Port bit P3A0 is used to command the conversion process in the ADC. A high to low transition initiates a conversion and a high STS of ADC indicates that ADC is busy. At the end of conversion, the STS line goes low. This transition is used to strobe the converted data into the latches at U40 and U41.

Input voltage range

This circuit can be operated with inputs of 0 to 10V, -5V to +5V or -10 to 10V. This is determined by setting the jumpers JP25 and JP24 as shown below.

Input Range	JP24	JP25
Unipolar 0 to 10V	2-3	1-2
Bipolar -5 to +5V	1-2	1-2
Bipolar -10V to +10V	1-2	2-3

An example program for using the analog to digital voltage converter is given Chapter 10. The user can use this program for developing custom applications.

6.9 ONBOARD 8-BIT D/A CONVERTER

ESA 86/88-3 features an onboard digital to analog converter using DAC 0800 IC at U36. The user may program the DAC to suit his needs using the interfacing information described here. For using the DAC, the user must connect +12V and -12V supplies to the trainer at the Power connector J3. The Analog output may be obtained from test-point J13 on the trainer. Interesting waveforms may be observed at this point by programming the DAC suitably.

The digital inputs to the DAC are provided through Port A of 8255-4 at U26 after initializing it as an Output port. The Analog output from the DAC is given to an operational amplifier (LM 318 at U24), which serves the purpose of current to voltage conversion. A 10K POT (VR1) is provided for offset balancing of the op-amp. The reference voltage needed for the DAC is obtained from an onboard voltage regulator μ A723 at U35. The voltage generated at this regulator is about 8 Volts. The output from the DAC varies between 0 and 5 Volts depending on the input digital pattern fed to the DAC.

The Control Register and Port addresses of 8255-4 are given in Section 6.1. An example program to generate triangular waveform using the DAC is given in the Chapter 10.

6.10 PARALLEL PRINTER INTERFACE

ESA 86/88-3 provides the user the facility to obtain a hard copy via Centronics compatible Parallel Printer Interface in Serial mode. The driver for this interface is any Centronics compatible parallel printer can be connected to ESA 86/88-3 using this feature. However to get properly formatted listings, it is recommended to use an 80/132-column printer.

6.10.1 INSTALLATION AND OPERATION

1. Switch OFF the power supply.
2. Configure the trainer for Serial Mode of operation and set the desired baud rate
3. Enable Printer Driver through on-board DIP SWITCH (SW6 = ON)
4. Connect the 25-Pin D type end of a Centronics cable to J7 ESA 86/88-3. The other end of connector must be connected to the printer.
5. Switch ON the power Supply. You should get the proper sign-on message as described in Chapter 2.
6. Now if the command prompt does not appear check the installation, the connector or Online/Ready indicator on the printer.
7. When the command prompt appears, the printer is assumed to be "attached" to ESA 86/88-3.
8. To "detach" the printer, put OFF SW6 of on-board DIP Switch and press RESET.

NOTE

- ♦ The Centronics connector cable could be obtained from ESA as an option.
- ♦ The users may build their own connector cable using the information given in section 6.10.5 and 6.10.6. However the cable must be short enough so that it does not load the 8255.



When the printer is attached, any character output on to the console is also output to the printer. For example, to obtain a hard copy of the contents of memory location in Hex, (byte format), use the command **D<addr1 >,<addr2><CR>**.

The entire contents of memory locations between the specified range are printed as it appears on the screen, including the command line.

NOTE: All control and invalid ASCII characters are printed as "."

Similarly, you can obtain a disassembled listing use the **Z** command when the printer driver is enabled.

6.10.2 THEORY OF OPERATION

The printer software makes use of 8255-3 at U1 whose Port B and Port C lines are brought out to 25-Pin D-type connector J7. This 8255 is initialized as given below, if the SW6 of DIP Switch is put ON.

PORT A	NOT USED
PORT B	O/P (Data port)
PORT C (Upper)	O/P
PORT C (Lower)	I/P

NOTE

- 1) The user can use this 8255 as general purpose I/O, when not using the printer by putting OFF the SW6 of DIP switch.
- 2) Handshaking is implemented using STROBE* and BUSY signals

6.10.3 ERROR MESSAGES

- ♦ If the Busy is not asserted the system will display the message

Printer Busy... Waiting

Now, the status of Busy line is kept under polling, until it is asserted. This process can be aborted by pressing <Ctrl C>. The system will then display the message

Printer aborted

- ♦ If no acknowledgement for data transfer is received from printer then a message

No ACK from Printer

is output. Please check the installation and connector once again in this case. Putting OFF SW6 of on-board DIP Switch and then pressing RESET will disable the printer.

6.10.4 PORT SPECIFICATIONS

The signals used for Parallel Printer Interface conform to the following specifications.

- ♦ Centronics Compatible
- ♦ Synchronization via STROBE* pulses
- ♦ Handshaking via BUSY signal
- ♦ Logic levels TTL Compatible



6.10.5 CONNECTOR DETAILS

Pin on 8255-3 at U1	Signal	Direction From ESA 86/88-3	Description	Pin No On Connector J7
13 (P3C4)	STROBE	O/P	STROBE* pulse to the printer	1
18 (P3B0)	Data 0	O/P		2
19 (P3B1)	Data 1	O/P	These signals	3
20 (P3B2)	Data 1	O/P	represent 8 bits	4
21 (P3B3)	Data 1	O/P	of parallel data	5
22 (P3B4)	Data 1	O/P	HIGH=1	6
23 (P3B5)	Data 1	O/P	LOW=0	7
24 (P3B6)	Data 1	O/P		8
25 (P3B7)	Data 1	O/P		9
17 (P3C3)	BUSY	I/P	A high indicates that the printer cannot receive data. ¹	11
7	GND		Signal ground	18 - 25

¹BUSY signal maybe high in following cases

- During data entry
- During Printing operation
- In OFF-LINE states
- During printer error status

6.10.6 DIRECT OUTPUT TO PRINTER

When the printer interface is enabled, any character sent to the console is sent to the printer also. This facility is available in the serial mode of operation only. However, user can access a routine "Print String" to send a string directly to the printer. This routine can be called from the user's program when the system is operating in either mode. Further, this routine prints the string regardless of the setting of SW6. The details of the routine "Print String" and a demonstration example are given in the Chapter 10.

6.11 ESA 86/88-3 CONNECTORS

A summary of all ESA 86/88-3 connectors is given below. Refer Appendix C to locate these connectors and Appendix B for their signal definitions.

- J1 & J2** 50 Pin Bus connectors connected to system bus.
- J3** 4 –terminal power connector.
- J4** 9 pin, D-type, female connector for RS 232C compatible serial communication.
- J5** 9 pin, D-type, male connector for RS 485 compatible serial communication.
- J6** PC Keyboard Connector.
- J7** 25- Pin Parallel Printer interface Connector.
- J8 & J9** 26-Pin parallel I/O Connectors.
- J10** 10-Pin Header for 8253 timer signals.
- J11** 15-Pin Flow-strip for LCD Module.
- J12** 9 Screw terminals for ADC inputs.
- J13** DAC output Test Point.



CHAPTER 7

ESA 86/88-3 EPROM PROGRAMMER SYSTEM

7.1 INTRODUCTION

ESA 86/88-3 Monitor program supports all the functions of ESA EPROM Programmer interface. Thus ESA 86/88-3 in conjunction with this interface module forms a powerful and easy to use EPROM Programmer system. This chapter describes the use of this EPROM Programmer system.

The system permits the user to Program, Verify Blank Check & Read any of the popular EPROMs - 2716 through 27512. The system hardware consists of the interface module connected to the Parallel I/O Port of the trainer using a 26-core ribbon connector. The software is invoked from the ESA 86/88-3 monitor itself.

The EPROM Programmer interface module provides a 28 pin ZIF socket for placing the EPROMs. When a 24-pin EPROM is to be placed, it must be aligned with the bottom row i.e. top two rows of ZIF socket are to be left blank. The system uses Intelligent Programming Algorithm whenever possible thus reducing the programming time significantly.

The devices supported by the system, their programming voltages and the type number to be entered by the user are listed below: The device selection is totally software-controlled. No hardware changes are necessary for working with any of these devices.

Device (EPROM)	Programming Voltage Vpp	Vpp Pin No. on ZIF Socket	Type Number to be entered
2716	25V	23	2716
2732 / 2732D	25V	22	2732
2732A	21V	22	732A
2764	21V	1	2764
27C64 / 2764A	12.5V	1	764A
27128	21V	1	0128
27128A / 27C128	12.5V	1	128A
27256 / 27C256	12.5V	1	0256
27256	21V	1	2256
27512 / 27C512	12.5V	22	0512

NOTE

1. ESA EPROM Programmer Interface is optionally available with ESA 86/88-3 and is not part of the default scope of supply. Please contact the manufacturer for further details.



2. The description given in this chapter hold good for either mode of trainer operation although the outputs of the commands are described as they would appear on the console in serial mode, while running the DOS driver package XT863.EXE. The only difference that would occur in the Stand-alone mode of operation would be in the output formatting on the LCD.

7.2 INSTALLATION

1. Turn OFF power to ESA 86/88-3 trainer.
2. Attach the hardware module to ESA 86/88-3 over the connector J8 using the 26-core ribbon cable supplied with the module.
3. Connect the black, yellow and blue wires coming from the 4-pin polarized connector on the programmer module to the corresponding power supplies as shown below

Color of the wire	Supply to be connected
BLACK	GND
YELLOW	+12V
BLUE	+30V

4. Power ON the system.

CAUTION: The following precautions must be taken failing which, the device in use as well as the PROM Programmer system is liable to be damaged.

- i. Spurious application of programming voltage to EPROM during switching operations with ESA 86/88-3 may damage the EPROM. To avoid this, insertion/removal of EPROM should be done only when the trainer is powered ON.
- ii. Ensure that the device type number entered corresponds to the EPROM inserted in ZIF socket.
- iii. Avoid attempting to program already damaged devices.

7.3 OPERATION

The EPROM programmer can be invoked either from the serial or the stand-alone monitor. The following discussion holds good for operation with the EPROM Programmer system for both modes of trainer operation except wherever indicated.

Enter P when prompted for command entry. Now, the EPROM programmer software gains control of the system and it will display a list of EPROM Programmer commands as follows.

P: Program R: Read B: Blank Check V: Verify E: Exit to Monitor

Select Option :

This is the command prompt of EPROM Programmer system for ESA 86/88-3 Trainer. Enter either of the following characters to proceed with the appropriate EPROM Programmer function (P/R/B/V). Enter 'E' to exit to the System Monitor.



The parameters required for each of the above commands are described in the following sections. For all these commands the system shows the default values. The user may modify these parameters to suit individual requirements. To abort any command the user can enter <Esc>; then the system will return to the EPROM Programmer command prompt.

Note: All commands of ESA 86/88-3 EPROM Programmer System except Blank Check use the trainer memory as buffer space. Thus it is advisable **NOT** to use the Monitor's stack area (i.e. locations 0:0 to 0:1FFFH) for operations with this interface as it may lead to unpredictable results.

7.3.1 P (PROGRAM) command

This command is used to program an EPROM. Enter P at the EPROM Programmer command prompt to execute this command. This command requires the following four parameters:

- PROM Type** : EPROM Type (One of the types listed in section 7.1)
- From Buffer(E/O/A):** This parameter specifies if Odd, Even or All locations from the buffer are to be programmed into the EPROM
- Buffer Start** :
 Segment : Segment address of the starting location of the buffer. The default value will be the current value of CS register
 Offset : Offset from the starting address of the current segment. The default value will be the current value of IP register
- Buffer End*** : Ending address of the buffer. Default value of this parameter is the maximum buffer offset value calculated using the device size and the user specification if Odd, Even or All locations from the buffer are being programmed.
- PROM Start** : Absolute starting address of the EPROM
(from where programming is to begin)

As soon as 'P' is typed, the system will display the current values for the above parameters. The user may enter new parameter values value followed by <CR> or simply enter <CR> if the displayed value is not to be changed.

In case of any invalid entry, the system displays an error message **"Invalid Parameters"** and prompts for the parameter again. Entering <Esc> at any stage will cause the system to return to PROM Programmer prompt.

Note that the parameters must satisfy certain conditions as listed below.

- i) PROM type should only be one of the types listed in section 7.1.
- ii) If only Even locations of the buffer are being used, both Buffer Start and Buffer End offset addresses must be even addresses.
- iii) If only Odd locations of the buffer are being used, both Buffer Start and Buffer End offset addresses must be odd addresses.



- iv) Buffer end address can be only an offset (Segment is not allowed). The segment value is the one specified for the Buffer start address.
- v) Buffer end address must be greater than or equal to the offset value of the Buffer start address.
- vi) The EPROM must have enough space to accommodate all the bytes specified by the Buffer start address and Buffer end address.

NOTE: 8086-memory space consists of an even bank and an odd bank. Thus, when an EPROM is programmed from alternate source bytes i.e. bytes from successive even or odd addressed, only the even or the odd strip is used. Although 8088-memory space is linear, it is organized as even and odd arrays on ESA 86/88-3.

***CALCULATION OF BUFFER END ADDRESS OFFSET**

The Buffer end address offset is calculated using the following relation.

If **ALL** locations from the buffer range are to be programmed into the device, then the default Buffer End Address offset = Buffet Start Offset + Max. Absolute address of the device (i.e. device size) or FFFF, whichever is smaller.

If only **EVEN** locations from the buffer range are to be programmed into the device, then the default Buffer End Address offset = Buffet Start Offset + 2 x Max. Absolute address of the device or FFFE, whichever is smaller. This is applicable if the Buffer Start address is an even location, else the monitor adds 1 to the default Buffer End Address offset.

If only **ODD** locations from the buffer range are to be programmed into the device, then the default Buffer End Address offset = Buffet Start Offset + 2 x Max. Absolute address of the device or FFFF, whichever is smaller. This is applicable if the Buffer Start address is an odd location, else the monitor adds 1 to the default Buffer End Address offset.

Example: For EPROM type 2764, the highest absolute address is 1FFFH. Then, the following table illustrates the Buffer end address calculation procedure in different cases. All calculations are made using HEX values only.

Buffer Start address offset	Locations from Buffer	Default Buffer End address offset
0	ALL	$0 + 1FFF = 1FFF$
1000	EVEN	$1000 + 2 \times 1FFF = 3FFE$
1000	ODD	$1000 + 2 \times 1FFF + 1 = 3FFF$
F000	ALL	$F000 + 2 \times 1FFF = 12FFE (> FFFF)$ Hence value = FFFF

After correct entry of the parameter values by the user, the system checks the EPROM for blank value (0FFH) for the specified address range. During this time, the message **Blank Checking.....** is displayed.



If the EPROM is not blank, the following prompt appears:

PROM not blank !!!
Non-blank at – XXXX

where XXXX is the address of the first non-blank value found in the device.

If the EPROM is already blank, following message appears:

PROM is blank ...
Blank Check completed

Then the system proceeds with Programming the device with the data specified by the buffer limits and will display

Programming.....

The system also verifies the programmed data on a byte by byte basis. Intelligent Programming Algorithm is used if the EPROM can support it resulting in a considerable reduction of programming time.

If complete programming is successful, the system will display the following message, where XXXX is a 16-bit checksum and control will return to the EPROM programmer command fetch routine.

Programming Completed
Check Sum... XXXX

If programming is unsuccessful, the following information is displayed.

Failed At -- XXXX

where XXXX indicates the location in HEX where programming was unsuccessful. The system then returns to EPROM Programmer command prompt.

EXAMPLE: Enter P at the ESA 86/88-3 Monitor Command prompt

.P<CR>

P: Program R: Read B: Blank Check V: Verify E: Exit to Monitor

Select Option : P<CR>

PROM Type : 2716-2764<CR>

From Buffer(E/O/A): E<CR>

Buffer Start :
Segment : 0000-<CR>
Offset : 0000-2000<CR>



Buffer End : 5FFE-2FFF<CR>
Invalid Parameters

This is because the Buffer end address is an odd location, while only the even bank of memory is selected for use.

Buffer End : 5FFE-2FFE<CR>
PROM Start : 0000-2000<CR>
Invalid Parameters

This error occurred because the EPROM does not have enough locations to be programmed as that indicated by the Buffer range.

PROM Start : 0000-1800<CR>

Blank Checking.....

PROM is blank ...
Blank Check completed

Programming.....

Programming Completed
Check Sum... 0FC2

Now the programming is complete and the system returns to EPROM Programmer command prompt.

7.3.2 R (READ) command

This command is used to transfer the contents of the EPROM into the ESA 86/88-3 memory space. Enter 'R' to execute this command. This command requires the following parameters.

PROM Type : EPROM Type (One of the types listed in section 7.1)

To Buffer(E/O/A) : This parameter specifies if the data read from the device is to be loaded in the Even, Odd or All locations from the buffer.

PROM Start : Absolute starting address of the EPROM
(from where programming is to begin)

Buffer Start :
Segment : Segment address of the starting location of the buffer. The default value will be the current value of CS register

Offset : Offset from the starting address of the current segment. The default value will be the current value of IP register



In case of any invalid entry, the system displays an error message **“Invalid Parameters”** and prompts for the parameter again.

The parameters must satisfy the following relations. These relations hold good for Blank Check command also.

i) PROM Start $\leq$ Absolute end address of the EPROM

ii) PROM End $\leq$ Absolute end address of the EPROM and $\geq$ PROM Start.

If all parameters are entered correctly, the system proceeds with reading the device and displays

Reading.....

The EPROM data is transferred into ESA 86/88-3 memory, starting from the location specified as Buffer Start address into successive odd or even locations. Once all the bytes specified in the EPROM range are read the system displays a 16-bit Check Sum of the data read and indicates whether reading is completed.

Check Sum... XXXX

Reading Completed.

If however, all the bytes are not transferred into ESA 86/88-3 memory space, then the system displays the number of bytes read from the EPROM (in Hex) and a 16-bit check sum of these bytes (NNNN) as shown below. Such a situation is encountered when the buffer range specified is lesser than the EPROM range. Note that in this case the message **‘Reading Completed.’** is not displayed either on the console or on the LCD.

NNNN Bytes Read.

Check Sum... XXXX

The system then returns to the EPROM Programmer command prompt.

Also, if after reading, the bytes cannot be loaded in the buffer area, then the following message is displayed. This may happen if the buffer range selected is pointing to non-existent memory or ROM area

Memory Write Failure At – Seg:Off

where **Seg:Off** points to the first location of the buffer where writing the data read from the device failed. Then the system returns to EPROM Programmer command prompt.

Note: Selecting the ESA 86/88-3 Monitor stack location i.e. locations 0:0 to 0:1FFFH the buffer may lead to unpredictable results.

7.3.3 B (BLANK CHECK) Command

This command is used to check if a specified range in the EPROM is blank (contains 0FFH). Enter **‘B’** to execute this command. This command requires the following parameters.



PROM Type : EPROM Type (One of the types listed in section 7.1)

PROM Start : Starting address of the EPROM, from where Blank checking is to begin.

PROM End : Ending address of the EPROM, till which Blank checking is to be done.

If all the parameters entered are correct, the system proceeds with reading the data from the device and comparing it with the device blank state value (FF in case of EPROMs.) During this time the following message is displayed.

Blank Checking.....

If all the locations are blank then the following message is displayed, and control returns to EPROM Programmer command prompt.

PROM is blank ...
Blank Check completed

If any memory location in the specified range is not blank, the corresponding message and the absolute EPROM address of the first non-blank location is displayed as shown below.

PROM not blank !!!
Non-blank at – XXXX

where XXXX is the address of the first non-blank value found in the device. Note that only the first non-blank location address is displayed. Subsequent locations may or may not be blank. The system then returns to the EPROM Programmer command prompt.

7.3.4 V (VERIFY) command

This command is used to verify the contents of an EPROM against a source. Enter ‘V’ to execute this command. The parameters and their interpretation are completely similar to that with respect to ‘P’ Command.

PROM Type : EPROM Type (One of the types listed in section 7.1)

From Buffer(E/O/A): This parameter specifies if Odd, Even or All locations from the buffer are to be verified with the EPROM data.

Buffer Start :

Segment : Segment address of the starting location of the buffer. The default value will be the current value of CS register

Offset : Offset from the starting address of the current segment. The default value will be the current value of IP register

Buffer End : Ending address of the buffer. Default value of this parameter is calculated similarly as done with the ‘P’ command (explained in Section 7.3.1.)

PROM Start : Absolute starting address of the EPROM, from where verification of the device should begin)



In case of any invalid entry, the system displays an error message **“Invalid Parameters”** and prompts for the parameter again. When all valid parameters are entered the following message is displayed on the console.

Verifying.....

If the verification is successful, the appropriate message along with a 16-bit checksum of the data verified is displayed as follows.

Verify Completed.

Check Sum... XXXX

If the verification fails, a corresponding message is displayed as follows, indicating the first mismatch of data between the buffer and the EPROM.

Verify Failed At -- XXXX
PROM Data -- XX
Buffer Data -- XX

where XXXX and XX are address and data in Hex format respectively. The system then returns to the EPROM Programmer command prompt.

EXAMPLE: Verifying the first 1000 bytes of 27128 with the buffer contents. Enter P at ESA 86/88-3 Monitor command prompt to enter programmer

.P<CR>

P: Program R: Read B: Blank Check V: Verify E: Exit to Monitor

Select Option : V<CR>

PROM Type : 2716-0128<CR>

From Buffer(E/O/A): A<CR>

Buffer Start :
Segment : 0000-F000<CR>
Offset : 0000-<CR>

Buffer End : 3FFF-0FFF<CR>

PROM Start : 0000-<CR>

Verifying....

Verify Failed At -- 0000
PROM Data -- 02
Buffer Data -- FA



The output shows that the first PROM location where the data from the device is not identical to the data in the corresponding buffer location is 0000. The system now returns to the EPROM Programmer command prompt.

7.3.5 E (EXIT) command

This command is used to terminate the EPROM Programming software and return control to the ESA 86/88-3 monitor. The monitor sign-on message then appears on the console or the LCD followed by the command prompt on the next line.



CHAPTER 8

ESA 86/88-3 SYMBOLIC ONE-LINE ASSEMBLER

ESA 86/88-3 provides a very convenient environment for Assembly Level Programming in either mode of trainer operation. This chapter describes the use of the powerful monitor resident ESA 86/88-3 Symbolic One-Line Assembler along with the other programming facilities viz. Label Commands and Disassembly. Appendix F contains a list of 8086/8088 CPU instructions supported by this assembler. Note that this assembler does not support 8087 NDP instructions.

8.1 ESA 86/88-3 SYMBOLIC ONE-LINE ASSEMBLER

The monitor resident Symbolic One-Line Assembler provided with ESA 86/88-3 Trainer is capable of translating the Mnemonic Instruction codes to equivalent machine codes for the 8086/8088 CPU. The translated code is immediately loaded into appropriate memory locations. The assembler supports the standard 8086/8088 mnemonic syntaxes with some minor and easy to understand modifications. This assembler supports labels (Symbolic references) also, that can have a maximum length of three significant characters. However a label can be referenced only if it has already been defined i.e. the assembler supports backward references only.

In addition to the standard instruction mnemonics, ESA 86/88-3 Symbolic One-Line Assembler supports some useful Assembler directives (Pseudo op-codes). These directives can be used to set the origin, define symbolic constants, initialize byte, word and string values, and to reserve memory space.

8.2 ASSEMBLY COMMAND (A)

8.2.1 FORMAT AND OPERATION

The command syntax for invoking ESA 86/88-3 Symbolic One-Line Assembler is

A[<address>]<CR>

The Assembly command has a single optional command parameter, which is the assembly address for source instructions. When prompted for command entry, enter 'A' followed by the assembly address and <CR>. Now the monitor will transfer control to the Assembler and the user can enter any of the 8086/8088 CPU instructions or directives supported by this Assembler. These commands and directives are described in detail in later sections.

NOTES

1. The assembly address is optional and if entered, the Assembler will start assembling source statements from this address. In case, no address is entered the assembler will use the current contents of the CS and IP registers as the effective address for assembly.



2. While using the Assembler, the user must enter source statements/programs only in the User RAM area starting from location 2000H. Entering program in reserved locations may lead to unpredictable results.
3. Other ESA 86/88-3 monitor commands cannot be executed while the Assembler is active.

8.2.2 ASSEMBLER SYNTAX DESCRIPTION

Once the assembler is invoked, the system will output the assembly address in **Segment:Offset** format followed by the instruction code and disassembled instruction at this address. On the next line, the assembler will display the assembly prompts **A:** and wait for the user to enter a source statement.

Now the user can either enter a source statement, carriage return if the current instruction is not to be changed or the character '!' and <CR> to terminate the command

If the user enters a new source statement, it is immediately translated and the machine code generated is stored at the appropriate memory locations. If any errors are detected or if the statement is written at an invalid location (viz. EPROM Area or non-existent memory), appropriate error message is issued next line and location counter is not updated and the old line is displayed again. If there are no errors, the assembled machine code is immediately loaded into the memory; the valid source statement and its machine code will overwrite the previous mnemonic and its corresponding machine code while updating the location counter. Now the next location counter (address) is displayed followed by the instruction code and disassembled instruction at this address. This is followed by the assembly prompt **A:** on the next line

NOTES

- ♦ The source line entered by the user is analyzed only after a carriage return is entered. Thus user can correct the entry errors using the Backspace key.
- ♦ The address displayed when the assembler is invoked will bear the current CS and IP register contents unless the user specifies a different address with the 'A' command.
- ♦ If the user doesn't wish to change the displayed instruction he can enter the carriage return. The location counter will be updated; the new value of the location counter is displayed along with the machine code, the disassembled instruction at that location and the assembly prompt on the next line. The process continues thereafter.
- ♦ If the user enters '!' followed by <CR> the assembler is terminated and control returns to the ESA 86/88-3 Monitor command prompt.

EXAMPLES

1. Invoking the ESA 86/88-3 Symbolic One-Line Assembler at the current PC value.

```
.A<CR>
0000:2000 FE C0 INC    AL
A:
```



Here, the CS & IP register contents before invoking the Assembler are assumed to be 0000H & 2000H respectively. The display format shown is for the serial mode of operation. The location counter is followed by a string of the machine code and the current instruction at that address. Further, the system will output the Assembly prompt **A:** in the next line.

In the stand-alone mode the display format on the LCD is slightly different and is as given below. In this format, the display string following the location counter is the current instruction at that address. This is followed by the machine code for that instruction and the assembly prompt **A:** on the next line.

```
.A<CR>
0000:2000
INC  AL
FE C0
A:
```

2. Invoking the ESA 86/88-3 Symbolic One-Line Assembler with a specific address.

```
.A 0:2500<CR>
0000:2500 8B C3 MOV  AX,BX
A:
```

The output on the LCD in case of stand-alone mode of operation will be as follows.

```
.A 0:2500<CR>
0000:2500
MOV  AX,BX
8B C3
A:
```

3. Writing a small program using ESA 86/88-3 Symbolic One-Line Assembler. This example illustrates the use of the assembler in Serial Mode to write a small program. Some common errors are also described in the example and explanation pertaining to these errors and display format is given at the end of the example.

```
.A<CR>
0000:0000 00 00 ADD  [BX] [SI],AL
A:  ORG 2000<CR>
0000:2000 E0 E9 LOOPNE 1FEB
A:  MOV AX,AABB<CR>
Invalid Operands.
0000:2000 E0 E9 LOOPNE 1FEB
A:  MOV AX,0AABB<CR>
->B8 BB AA
0000:2003 B0 90 MOV  AL,90
A:  MOV BX,2222<CR>
->B8 22 22
0000:2006 03 C3 ADD  AX,BX
A:  <CR>
0000:2008 50  PUSH  AX
```



```

A: 1VAL: DB 10
Labels should start with Alphabet.
0000:2008 50      PUSH      AX
A:  VALUE: DB 10<CR>
    ->DB 10
0000:2009 FB      STI
A:  MOV CL,@VAL <CR>
    ->8A 0E 06 20
0000:200D 26      ES
A:  UP INC BX<CR>
Invalid Mnemonic.
0000:200D 26      ES
A:  UP: INC BX<CR>
    ->43
0000:200E C3      RET
A:  INC [SI]<CR>
Illegal Operands.
0000:200E C3      RET
A:  INCW [SI]<CR>
    ->FF 04
0000:2010 34 01  XCHG      AL,01
A:  JMP UP<CR>
    ->E2 FB
0000:2012 CC      INT      03
A:  <CR>
0000:2011 EE      OUT      DX,AL
A:  !
.

```

NOTES:

1. The Assembler is invoked without any specific address. Hence it takes the current CS & IP register contents as reference memory locations.
2. Use of Origin Control Directive (ORG) is made here before entering any instructions since the current location counter does not point to user RAM area. The location counter is now modified to point to the address specified by this directive.
3. An error message follows the first instruction entered here, since the Hex operand does not begin with a numeric. The location counter is not updated until a valid source statement is entered. In such case, the appropriate error message is displayed and the current location counter is displayed again followed by the machine code and instruction at that address and the assembly prompt **A:** on the next line. The Mnemonics syntax for ESA 86/88-3 Symbolic One-Line Assembler is described in detail in a later section.
4. When an instruction is successfully assembled, the machine code for the instruction is output on the next line as shown in the third instruction entered in the above example. This is preceded with a -> symbol and the location counter is suitably updated and subsequently displayed with the Assembly prompt.



5. If the user does not wish to change the current instruction at a location, then entering a **<CR>** at that location will maintain the instruction there and update the location counter. Note that in this case, the current line is overwritten by the next location counter and the machine code and instruction at this address. Underlined lines in the above example indicate that they are overwritten by the next output line. This is illustrated in the instruction at locations 0:2006H and 0:2012H
 6. Labels supported by the assembler can have a maximum of three significant characters. If the label contains more than three characters, then they can be referred using only the first three characters. The ESA 86/88-3 Monitor supports some additional label commands that are discussed later. Also refer Section 8.3.5 for more details on the usage of labels.
 7. Almost all instructions mnemonics may be 'optionally' suffixed with the letter 'W' or 'B' to indicate Word or Byte operation. **However, for indexed addressing mode instructions that have only one operand pointing to the effective address, this suffix becomes mandatory for the correct assembly of the instruction.** This is equivalent to specifying **Word Pointer** or **Byte Pointer** with the mnemonic, as is the case with some other assemblers like MASM, TASM, 2500AD or Microsoft Debug.
- Example:** For the instruction **INC [BX][SI]1234**, the source statement to be entered during assembly will be **INCW [BX][SI]1234** for Word operation. If Byte operation is intended, then the source statement will be **INC [BX][SI]1234**.
8. The user can terminate the active assembler by entering '**! <CR>**' at the Assembly prompt **A:** The user will now be able to use other ESA 86/88-3 Monitor commands.

Using ESA 86/88-3 Symbolic One-Line Assembler in Stand-alone Mode of operation:

In stand-alone mode of operation, the display format on the LCD will follow the same lines as that described in Examples 1 & 2. However some points must be noted here.

- ♦ The Assembly prompt follows the location counter, the instruction at that address and the machine code in subsequent lines and generally occurs on the fourth line of the LCD.
- ♦ When an instruction is successfully assembled, the display is refreshed and the next location counter with the corresponding instruction and machine code is directly displayed. Thus the user may understand that a source statement has been successfully translated by watching the location counter.
- ♦ In case of an error, the error message is displayed after refreshing the display once. Now, the assembler waits for a user strobe viz. **<CR>** and then displays the same location counter with the respective parameters as explained before. Thus, the display pattern while entering the second source statement in the above example in stand-alone mode will be as described below.

```
0000:2000
LOOPNE 1FEB
E0 E9
A:MOV AX,AABB<CR>
```



The LCD is now refreshed once and will display the error message

Invalid Operands.

Now, the Assembler waits for a <CR> from the user to proceed with the assembly process. When the user enters <CR> the display is refreshed again and the same location counter with the corresponding parameters is output as follows.

```
0000:2000
LOOPNE 1FEB
E0 E9
A:MOV AX,0AABB<CR>
```

Upon successful assembly, the next location counter is directly displayed as shown below.

```
0000:2003
MOV AL,90
B0 90
A:
```

- ♦ If the user wishes to view the machine code generated for the instruction entered, then he can do so by using the Origin Control Directive (ORG) specifying the previous address in the statement.

For instance, the user can view the machine code generated in the previous example by following the pattern described below.

```
0000:2000
LOOPNE 1FEB
E0 E9
A:MOV AX,0AABB<CR>
```

```
0000:2003
MOV AL,90
B0 90
A:ORG 2000<CR>
```

```
0000:2000
MOV AX,AABB
B8 BB AA
A:
```

It may be seen from the above that ESA 86/88-3 provides an easy-to-use assembly level-programming environment even in the stand-alone mode. The user is urged to try out the above examples in both modes of trainer operation to get well acquainted with ESA 86/88-3 Symbolic One-Line Assembler. Other assembly support instructions viz. label commands and Disassembly, are discussed in subsequent sections.



NOTE: Before invoking ESA 86/88-3 Symbolic One-line Assembler for the first session it is advisable to use Label Clear command (*described later*), to clear all random label definitions in the symbol table.

8.3 ASSEMBLY LANGUAGE CONVENTIONS

The detailed syntax governing source statements is described in this section. Each line of a source statement can contain a Label field, a Mnemonic field & an Operand field in that order. Thus the general syntax for a source statement involving a CPU instruction will be

[Label] <Mnemonic> [operand(s)]

The different fields in a source line are identified by the order in which they appear. These fields are separated by one or more blank spaces or a comma acting as the delimiters.

8.3.1 LABEL FIELD

The label field is not exclusive in nature and may be overwritten by mnemonics or assembler directives that are part of the source statement. A label can have a maximum of three significant characters. The first label character must be an alphabetic character. The following characters if present can be alphanumeric. If more than three characters are assigned to the label, the assembler stores only the first three characters and ignores the others. Register names and assembler directives given in the list below form reserved words and cannot be used as labels.

AH, AL, ASC, AX, CS, DB, DS, DSP, DW, EQU, ORG, SS, ES, CH, CL, CX, BH, BL, BX, SP, BP, SI, DI, DH, DL, DX

8.3.2 MNEMONIC FIELD

The next field is the mnemonic field. This field can overwrite the Label field and always begins with an alphabetic character. This field is also used for entering Assembler Directives as source statements. The assembler supports the standard INTEL mnemonics for 8086/8088 CPU instructions, with some easy to understand modifications in a few cases.

The detailed syntax of Mnemonics and Addressing modes supported by ESA 86/88-3 Symbolic One-Line Assembler is given at the end of this chapter.

8.3.3 OPERAND FIELD

The operand field follows the label and Mnemonic fields. This field holds the operands specific to the instruction. An instruction may or may not have an operand at all. The operands can be register symbols, data, addresses or labels. This assembler supports all standard addressing modes of 8086/8088 CPU. Please refer to the Mnemonic syntax description, the discussion on labels and programming examples given in the manual to get familiar with the Mnemonic and operand syntax supported by this Assembler.

When a numerical value is entered in the operand field, the assembler accepts it in hexadecimal format only. Furthermore numbers should always start with a digit. Thus the value A5H should be entered as 0A5.

Labels can be used in the operand field, provided that they are been already defined.



8.3.4 SEGMENT OVERRIDE OPTION

Segment override option can be exercised by giving the segment register instruction prior to the actual instruction e.g. "MOV ES:[SI], AL" instruction specifies a byte movement with the segment register ES overriding the default segment register DS. To realize this instruction in the present assembler, the user must enter the following source statements.

```
ES <CR>
MOV [SI],AL
```

NOTE

The segment register override instruction will be applicable for all the instructions that follow it. The assembler will not check for the override capability of the next instruction. Thus if you want the override option for only one instruction, you must restore the default segment register explicitly.

8.3.5 MORE ON USAGE OF LABELS

Whenever labels are included in source statements, the assembler understands them as address references. The following discussion on labels should serve as guidelines during program assembly using ESA 86/88-3 Symbolic One-line Assembler.

- ♦ The label definitions can have a maximum of three significant characters.
- ♦ If a previously defined label is defined again during assembly, then the new definition is also recorded in the symbol table. All future references to this label will result in only the new address being referred.
- ♦ More than one label can be defined at one location. In such cases, this location can be referred by using any of these labels.
- ♦ Labels can be used with Assembler directives also. However, if used with ORG directive, there is no label assignment in the symbol table since this directive does not perform any memory allocation.
- ♦ Label referencing for different addressing modes can be done using the label name directly with the instruction.

EXAMPLES (Refer Section 8.5 also)

1. The instruction **MOV AX,R1** will move the address location referenced by R1. If R1 is defined at 0:2500H, then executing this instruction will transfer 2500H to AX register.
2. The instruction **MOV AX,@R1** will move the contents at the location referenced by R1.
3. The source statement **ORG R1** will direct the assembler to begin assembly from location referenced by R1
4. The same holds true for branching instructions viz. LOOP and JUMP.



8.4 LABEL COMMANDS

ESA 86/88-3 Monitor provides two support commands for the assembler pertaining to usage of labels. The commands are briefly discussed here.

1. LABEL CLEAR (LC)

This command clears all labels from the symbol table in the system memory.

FORMAT & OPERATION

LC<CR>

To use this command in either mode of operation, the user has to enter **LC** followed by a **<CR>**.

This command will clear the symbol table and all label definitions generated in the earlier invocations of the assembler are lost. If this command is not used, earlier label definitions will remain valid for this session and until the command is used.

It is urged that this function be used whenever a fresh session with the trainer is begun especially before invoking the assembler for the first time. This function will clear all random label assignments.

2. LIST LABELS (LL)

This command outputs the total count of label definitions in the symbol table and lists them.

FORMAT & OPERATION

LL<CR>

To use this command in either mode of operation, enter **LL** followed by a **<CR>**.

This command will first output the Label Count, which is a hexadecimal count of the number of labels defined in the symbol table. The symbol table follows the FIRST IN FIRST OUT method of storing label definitions and hence, the order of label list is independent of address sequence. The labels are listed in the order that they are defined.

If there are no labels defined in the symbol table, this command will output a Label Count of zero with the message '**No Labels Defined.**' Thus this command can be used to check the validity of the Label Clear command also.

If the same label name or symbol is defined more than once the command will output all the symbols with their addresses.

EXAMPLES

1. Use of List Labels command when no labels are defined.

```
.LL<CR>
Label Count: 0000
No Labels Defined.
.
```



This example shows the output for the LL command in the serial mode when there are no labels defined in the symbol table. In the stand-alone mode of operation, the monitor refreshes the LCD before displaying the label count and the message.

2. Use of List Labels command in stand-alone mode of operation

.LL<CR>

(The monitor first refreshes the LCD and then displays the Label Count).

.Label Count:0006

(The monitor now waits for a user strobe viz.<CR> before listing the labels defined).

```
<CR>
0000:2000  R1
0000:2005  BAC
0000:200A  L1
0000:2002  R2
<CR>
0000:3000  L2
0000:3000  UP
.
```

This example illustrated the use of LL command in stand-alone mode of operation. The listing format includes the address location of the label followed by the symbol (Label Name) defined. This format is the same for output in serial mode also.

Note that after listing four consecutive label definitions, the monitor waits for a user strobe viz.<CR> before proceeding with the output. In the serial mode the monitor waits after listing 25 label definitions and then displays a message 'Press Any Key.' The listing continues upon pressing any key. In either mode of operation, entering <ESC> key while the monitor is waiting terminates the command and the monitor returns to command prompt mode.

8.5 ASSEMBLER DIRECTIVES (PSEUDO OP-CODES)

In addition to the normal op-codes that generate executable machine instructions the Assembler recognizes some Pseudo op-codes which occupy the mnemonic field like normal op-codes. These directives instruct the assembler to perform certain functions like setting the origin, defining symbolic constants, initializing byte, word or string values, etc. ESA 86/88-3 Symbolic One-line Assembler supports the following assembler directives.

- ♦ Origin control (ORG)
- ♦ Symbolic constant definition (EQU)
- ♦ Byte Initialization (DB)
- ♦ Word Initialization (DW)
- ♦ String constant initialization (ASC)

Each of these directives is now discussed briefly.



1. Origin control (ORG)

ORG <Value> <CR>

The location counter or address can be set to a specific value with the help of this directive.

It is generally used as the first program entry for the starting location of the assembled code. For example **ORG 2000** will result in the loading of next instruction assembled to start from location 2000H. The value to be specified with this directive can be entered in the **Segment:Offset** format also.

Example:

```
.A<CR>
0000:0000 27          DAA
A:   ORG 2000<CR>
0000:2000 00 3F      ADD [BX],BH
A:   ORG 0F000:0<CR>
F000:0000 FA        CLI
A:
```

This example illustrates the use of ORG directive to set location counters. After invoking the assembler, the location counter corresponds to the current content of the CS and IP registers. The first ORG directive is made to change the offset of the location counter only and hence no segment specification is made. In the second source statement ORG is used with segment as well as offset specification. This statement modifies both the segment and the offset of the location counter accordingly.

2. Symbolic Constant Definition (EQU)

User can define a value for a symbolic constant by using the 'equate' (EQU) directive

<Symbol> EQU <defined constant> <CR>

For example, the symbol T1 can be defined to be equal to 2000H, by entering **T1 EQU 2000**

Further the EQU directive allows the user to assign the value of another symbol. For example, if T1 is already defined as 2000H, then entering **T2 EQU T1** defines T2 also as 2000 H

```
T1 EQU 2000          ; T1 represents 2000H
T2 EQU T1            ; T2 represents 2000H
Y EQU 10             ; Y represents 10H
```

When using the symbols defined by this directive, the symbol name can be used directly with the instruction if the value to which it is assigned is required. If, however a reference to the location pointed by the symbol is required, then it can be done by using the character '@' before the symbol name.



Example: With the symbols defined above, the instruction **MOV BX,T1** will transfer the data word 2000H to BX register while the instruction **MOVW BX,@T1** will move the data word indexed by CS:2000H to BX register.

3. Byte Initialization (DB)

[<label>] DB <value> [, <value>] <CR>

The user can initialize a memory location to a particular value by the use of this command. This facility is particularly useful when entering a table of data as part of a program. This directive supports up to two constants separated by a comma or space.

E.g.

```
0000:2240 44      INC SP
A:      T1 DB 12,0A4<CR>
->DB 12,0A4
```

This source statement will initialize location 0:2240 with the value 12H and location 0:2241 with the value A4H. Further the symbol T1 is made equal to 2240H.

4. Word Initialization (DW)

[<label>] DW <value> <CR>

This directive is similar to DB directive except that with this directive, a word of memory rather than a byte is initialized. Further, only one value is permitted with one statement.

E.g.

```
0000:2240 44      INC SP
A:      TY DW 1240 <CR>
->DW 1240
```

will initialize the word location at 0:2240H with the value 1240H.

5. String constant Initialization (ASC) ASC 'String '

This directive allows the user to enter a string of characters and have these characters translated to ASCII Codes and stored in the memory. The string can have a maximum of 160 characters. The string must be enclosed within a pair of single quote (') characters.

E.g.

```
0000:2000 90      XCHG AX,AX
A:      ASC `ESA' <CR>
-> ASC `ESA'
```

This source statement will initialize locations 0:2000H, 2001H & 2002H respectively with 45H, 53H and 41H. Now the assembler will overwrite the original line, with the string displayed in opcode field.



8.6 DISASSEMBLY COMMAND (Z)

ESA 86/88-3 Monitor provides an extremely useful command viz. Disassembly in support of the Symbolic One-Line Assembler. With the help of this command the user can disassemble a range of memory locations into corresponding Assembly Mnemonics. Thus the user can view the program and subsequently verify it even if machine code has been entered directly.

FORMAT

Z[<start address>[,<end address>]]<CR>

OPERATION

- ♦ The command syntax for Disassemble command is similar for either mode of trainer operation.
- ♦ To use this function, enter **Z** when prompted for command entry. The user may now enter a <CR> to view the disassembled instruction corresponding to the current CS and IP register contents.

If the user wishes to view the instruction code at only a particular memory location, then that address must be entered followed by a <CR>.

To view the disassembled instruction Mnemonics for a range of locations, the user must specify the same by entering the start and end addresses of the range separated by a comma. Upon entering <CR> now, the monitor will output the disassembled instructions in the following format.

In Serial Mode, the following pattern will appear on the console

Segment:Offset	Machine Code	Mnemonic
-----------------------	---------------------	-----------------

In stand-alone mode operation the same parameters will appear on the LCD in the following pattern.

Segment:Offset
Mnemonic
Machine Code

After displaying the disassembled code, the monitor returns to command entry mode.

- ♦ The end address specified with the command is always an offset relative to the segment address specified with the start address. Hence a segment address should not be specified with the end address. However if no segment address is specified with the start address parameter, then the address range of disassembly is relative to current CS register contents.
- ♦ In stand-alone mode, the monitor waits for a user strobe after displaying one location, the instruction and the machine code at that location and then either returns to command prompt or refreshes the LCD and displays the next disassembled location, the instruction and the respective code. In serial mode, the entire specified memory range is disassembled and displayed as a continuous list after which the monitor returns to command prompt.



- ♦ In serial mode operation, display control commands viz. <Ctrl-S>, <Ctrl-Q> and <Ctrl-S> can be used to control the output flow to the console. (Refer Section 4.4)
- ♦ The labels or symbols defined during assembly are not displayed during the disassembly, Appropriate address references or values defined using symbolic constants replace the labels and symbols defined.

EXAMPLES

1. Disassembling the Hex codes at location 0000:2000H. Assume CS content is previously set to 0000 in Serial Mode of operation.

```
.Z 2000<CR>
0000:2000 92                XCHG AX,DX
.
```

2. Disassembling a range of memory locations from 0200:0030H to 0200:0040H.

```
.Z 200:30,40<CR>
0200:0030 BA E7 FF        MOV    DX,FFE7
0200:0033 F7 D0          NOT     AX
0200:0035 92             XCHG   AX, DX
0200:0036 B9 01 00       MOV     CX,0001
0200:0039 D2 FB          SAR     BL,CL
0200:003B 9A 00 10 00 FB CALLS  FB00:1000
0200:0040 CC             INT     03
.
```

In stand-alone mode of operation, the output for the same command will be in the pattern described below. As mentioned earlier, the monitor waits for a user strobe after disassembling and displaying one location and its corresponding parameters.

```
.Z 200:30,40<CR>

0200:0030
MOV    DX,FFE7
BA E7 FF<CR>

0200:0033
NOT     AX
F7 D0<CR>

0200:0035
XCHG   AX,DX
92<CR>

0200:0036
MOV     CX,0001
B9 01 00<CR>
```



```

0200:0039
SAR    BL,CL
D2    FB
<CR>

0200:003B
CALLS  FB00:1000
9A 00 10 00 FB
<CR>

0200:0040
INT     03
CC
<CR>
.

```

- ♦ Since the end address offset specified with the command is relative to the segment value of the start address, this command can disassemble and display a maximum of 64K bytes of machine code in a single operation.
- ♦ Specifying an end address offset value lesser than the start address offset results in an error.

8.7 ADDRESSING MODES

ESA 86/88-3 Symbolic One-Line assembler supports all the addressing modes of 8086/8088. The syntax to be followed for the various addressing modes is summarized as follows. Please refer Appendix F for the list of Instruction mnemonics and the syntax supported by the assembler.

Base Registers	[BX] or [BP]
Index Registers	[SI] or [DI]
Base + Displacement	[BX], [SI], [DI] or [BP] followed by value or label.
Base + Index	[BX][SI], [BX][DI], [BP][SI] or [BP][DI]
Immediate data	VALUE or LABEL
Direct addressing	[SI], [DI], @VALUE or @LABEL
Displacement + Base + Index	[BX][SI]VALUE or [BX][SI]LABEL

NOTE

1. All numerical values are at hexadecimal base.
2. Label when used, must have been defined in previous instructions.
3. Mnemonics may be appended with a 'B' or 'W' to specify whether the operands are 'bytes' or 'words' respectively.



CHAPTER 9

ESA 86/88-3 MONITOR ROUTINES

ESA 86/88-3 Monitor provides the user with several useful routines in either mode of trainer operation that significantly simplifies the task of program development. This chapter gives a list of user-callable routines provided by ESA 86/88-3 Monitor that are accessible to the user with their calling address locations and a brief summary of their functions. Example programs using these routines are given in Chapter 10.

The outputs of some routines are independent of the mode of trainer operation while some routines provide input and output facilities only on the default system I/O for a particular mode of operation.

Routines listed in Section 9.1 are dependent on the mode of trainer operation. When used in stand-alone mode, the outputs will be displayed on the LCD, and in case of serial mode of operation, the corresponding outputs will appear on the console. Similarly any input parameters that are required should be entered by the terminal keyboard in case of serial mode and by the PC keyboard interfaced with the trainer during stand-alone mode operation.

Routines listed in Section 9.2 are independent of the mode of trainer operation and are useful in applications where independent control of the I/O devices is required.

9.1 MONITOR ROUTINES DEPENDENT ON OPERATING MODE

1. Name of routine: **OUT_CHAR**
Function: Outputs an ASCII stored in AL register character to the console in Serial Mode or to LCD in Stand-alone Mode
Calling address: **FE00:0000**
2. Name of routine: **OUT_STRING**
Function: Outputs a string of ASCII characters stored in memory. The string is indexed by ES:AX where ES is the segment value and AX is the offset address of the starting location of the string.
Calling address: **FE00:0013**
3. Name of routine: **OUT_CRLF**
Function: Outputs a carriage return and line feed (ASCII code = 0AH, 0DH) to the LCD or console.
Calling address: **FE00:0031**
4. Name of routine: **OUT_BLANK**
Function: Outputs a blank character (ASCII code = 20H) to the LCD or console.
Calling address: **FE00:0049**



5. Name of routine: **OUT_BYTE**
Function: Outputs a byte value stored in AL register to the LCD or to the console.
Calling address: **FE00:0052**
6. Name of routine: **OUT_WORD**
Function: Outputs a Word value stored in AX register to the LCD or to the console.
Calling address: **FE00:006A**
7. Name of routine: **OUT_ADDR**
Function: Outputs an address value in **Segment:Offset** format to the LCD or to the console. The segment value to be displayed is stored in ES register and the offset in BX register.
Calling address: **FE00:0082**
8. Name of routine: **GET_CHAR**
Function: Reads an ASCII character from the keyboard and stores its Hex equivalent in AL register.
Calling address: **FE00:00A9**
9. Name of routine: **GET_CHAR_E**
Function: Reads an ASCII character from the keyboard and displays the character on the LCD or console depending on the operating mode. The HEX equivalent is stored in AL register.
Calling address: **FE00:00B8**
10. Name of routine: **GET_BYTE**
Function: Reads a byte value from the keyboard and stores it in AL register.
Calling address: **FE00:00C7**
11. Name of routine: **GET_WORD**
Function: Reads a word value entered from the keyboard and stores it in AX register.
Calling address: **FE00:00E0**
12. Name of routine: **GET_ADDR**
Function: Reads an address value from the keyboard in **Segment:Offset** format. The segment value entered is stored in ES register while the offset is stored in BX register. If no segment value is entered the current CS register value will be stored in ES register.
Calling address: **FE00:00F3**
13. Name of routine: **GET_STRING**
Function: Reads a string of ASCII characters from the keyboard and stores their HEX equivalents in memory indexed by DS:SI. The current content of DS register is the Segment value and that of SI register is the Offset value of the starting location of the string being stored. The string is terminated by <CR>.
Calling address: **FE00:010E**



14. Name of routine: **VALID_HEX**
 Function: This routine checks if the value stored in AH register is a valid ASCII Hex value (30–39 / 41–46). Stores FFH in AL register if found valid, and 00 if found invalid.
 Calling address: **FE00:0123**
15. Name of routine: **HEX_ASCII**
 Function: This routine converts a valid ASCII Hex value (30–39 / 41–46) stored in AH register to its equivalent character (0–9 / A–F) and stores the result in AL register. Note that the routine will not convert any other values and that the previous contents of AL register are destroyed.
 Example: 41H will be converted to 0AH; similarly 33H will be converted to 03H.
 Calling address: **FE00:0131**
16. Name of routine: **ASCII_HEX**
 Function: This routine converts a valid Hex character (0–9 / A–F) stored in AH register to its equivalent ASCII Hex value (30–39 / 41–46) and stores the result in AL register. Note that the routine will not convert any other values and that the previous contents of AL register are destroyed.
 Example: 0AH will be converted to 41H; similarly 03H will be converted to 33H.
 Calling address: **FE00:0152**
17. Name of routine: **BCD_BIN**
 Function: This routine converts a valid BCD value (up to 99) stored in AH register to its equivalent binary value in AL register. Note that the previous contents of AL register are destroyed.
 Example: 99 will be converted to 63; similarly 10 will be converted to 0A.
 Calling address: **FE00:016B**
18. Name of routine: **BIN_BCD**
 Function: This routine unpacks a 8-bit binary number to three unpacked BCD digits and stores them in 3 consecutive locations indexed by DS:SI After the conversion, DS:SI will point to the least significant byte (digit).
 Example: FFH stored in AL register will be unpacked into 3 unpacked BCD digits 2, 5 and 5 and will be stored at locations 0:3000H, 3001H and 3002H respectively (assuming DS = 0 and SI = 3000H).
 If the number is less than 100 (64H), then the most significant digit is taken to be 0. For e.g. 63H (decimal = 99) will be unpacked as 0, 9 and 9.
 Calling address: **FE00:018B**

9.2 MONITOR ROUTINES INDEPENDENT OF OPERATING MODE

The following routines are useful for independent control of I/O devices connected to the trainer. These routines can thus be used regardless of the operating mode of ESA 86/88-3

1. Name of routine: **SER_SEND_STRING**
 Function: Outputs a string of ASCII characters stored in memory to the console. The string is indexed by ES:SI where ES is the segment value and SI is the offset address of the starting location of the string. The string is terminated by 00H



- Calling address: **FE00:01AF**
2. Name of routine: **LCD_SEND_STRING**
Function: Outputs a string of ASCII characters stored in memory to the LCD. The string is indexed by ES:SI where ES is the segment value and SI is the offset address of the starting location of the string. The string is terminated by 00H
Calling address: **FE00:01CE**
3. Name of routine: **CLR_LCD**
Function: Clears the LCD interfaced with the trainer in either mode of operation.
Calling address: **FE00:01ED**
4. Name of routine: **PRINT_STRING**
Function: Prints a string of ASCII characters stored in memory when a parallel printer is interfaced with the trainer. The string is indexed by DS:SI where DS is the segment value and SI is the offset address of the starting location of the string. The string is terminated by 00H. Note that this routine is used for direct output to the printer regardless of whether the printer driver is enabled by DIP Switch.
Calling address: **FE00:01F3**



CHAPTER 10

PROGRAMMING EXAMPLES

This chapter describes some programming examples that can be executed on the ESA 86/88-3 trainer. These examples range from fairly simple ones designed to illustrate the use of various instructions to some comprehensive examples designed to illustrate the use of monitor routines and demonstration examples for onboard peripherals. It is strongly urged that the user read this chapter carefully to be able to use ESA 86/88-3 efficiently.

The examples are presented in a format that makes it convenient for the user to enter programs using the monitor resident ESA 86/88-3 Symbolic One-line Assembler. The HEX equivalents of the instructions are also given in the examples as they appear during disassembly. Thus the user may use Substitute Memory commands and enter the HEX codes at the memory locations directly, if so desired.

NOTE: User area of RAM starts from 0:2000H and program entry or execution should not begin from an address within this area.

General instructions

1. Enter the programs in the trainer memory at the locations shown along with the program using ESA 86/88-3 Symbolic One-Line assembler.
2. Some programs may require look-up tables. The corresponding data may be entered at the appropriate locations by using 'DB', 'DW' or 'ASC' directives, or by using the Substitute Memory Commands.
3. Using the GO command execute the program at its starting location. In most cases, the control returns to the monitor after program execution (because of the Breakpoint instruction **INT 3**). It is important to properly terminate any user program, failing which the program data may be lost.
4. After observing the results of these programs, it is urged that the user try different variations of these programs to get familiar with the ESA 86/88-3 system
5. The actual disassembly of the programs will not contain some of the labels used; instead their reference locations will be displayed. However, the user may use these labels while assembly.

10.1 FAMILIARIZATION EXAMPLES

These examples are designed to familiarize the user with the operation of ESA 86/88-3 system.

Example 1: This program computes the average of given word values stored in memory. The computed average is also stored at a given memory location. Note that this program does not check for overflow while forming the sum of the data values.



ADDRESS	OBJECT CODE	LABELS	MNEMONICS	COMMENTS
0000:2000	B8 00 00		MOV AX,0000	;Initialize
0000:2003	8E D8		MOV DS,AX	;segment registers
0000:2005	BE 20 20		MOV SI,2020	
0000:2008	B9 05 00		MOV CX,05	;Load Count and
0000:200B	03 04	BAK:	ADD AX,[SI]	;add the words
0000:200D	46		INC SI	;sequentially
0000:200E	46		INC SI	
0000:200F	E2 FA		LOOP BAK	
0000:2011	B9 05 00		MOV CX,05	
0000:2014	F7 F1		DIV CX	;Divide Sum by
0000:2016	BE 30 20		MOV SI,2030	;count
0000:2019	89 04		MOV [SI],AX	;Store Computed
0000:201B	CC		INT 03	;average in memory
0000:201C			ORG 2020	;Data Words stored
0000:2020			DW 1000	;at 0:2020H
0000:2022			DW 2000	
0000:2024			DW 3000	
0000:2026			DW 4000	
0000:2028			DW 5000	

- ♦ This program will compute the average of 5 data words entered at locations 0:2020H onwards. The result is stored at memory location 0:2030H
- ♦ Examine the contents of the word location RESULT (2030H). For the entries shown in the program, the result will be 3000H.

Example 2: The following program exchanges two blocks of data stored in memory using the powerful string instructions of 8086/8088. This program exchanges 1FH bytes from 0:3000H and 0:3200H onwards. The user can use this program with necessary modifications to exchange data blocks of desired size between desired locations.

ADDRESS	OBJECT CODE	LABELS	MNEMONICS	COMMENTS
0000:2000	BB 00 30		MOV BX,3000	;Store index
0000:2003	BA 00 32		MOV DX,3200	;references in
0000:2006	B8 00 35		MOV AX,3500	;registers
0000:2009	8B F3		MOV SI,BX	;Set up 1st block
0000:200B	8B F8		MOV DI,AX	;as source & a temp
0000:200D	B9 20 00		MOV CX,20	;block as destination
0000:2010	FC		CLD	
0000:2011	F3		REP	
0000:2012	A4		MOVS	
0000:2013	8B F2		MOV SI,DX	;Set up 2nd block as
0000:2015	8B FB		MOV DI,BX	;source & 1st block as
0000:2017	B9 20 00		MOV CX,20	;Destination address
0000:201A	FC		CLD	
0000:201B	F3		REP	
0000:201C	A4		MOVS	
0000:201D	8B F0		MOV SI,AX	;Set up temp. block as
0000:201F	8B FA		MOV DI,DX	;source & 2nd block as
0000:2021	B9 20 00		MOV CX,20	;Destination address
0000:2024	FC		CLD	
0000:2025	F3		REP	
0000:2026	A4		MOVS	
0000:2027	CC		INT 03	



- ♦ Enter the above program beginning at location 0:2000H
- ♦ Enter the desired data in locations 0:3000 to 0:301FH and 0:3200H to 0:321FH
- ♦ Execute the program using GO command. Now the data block at 0:3000 would be moved to 0:3200h and the data block at 0:3200H to 0:3000H. This can be verified using display memory commands.

Example 3: The following program converts a hexadecimal byte value to its ASCII notation. The example illustrates the use of the powerful translate (XLAT) and rotate instructions. The program assumes that the hex value is in AL register. The resulting ASCII representation is left in the AX register. Enter the program at 0:2000H and enter the required HEX value to be converted in AL register.

ADDRESS	OBJECT CODE	LABELS	MNEMONICS	COMMENTS
0000:2000	BF 12 20		MOV DI,2012	;Get address of ASCII
0000:2003	8D 1D		LEA BX,[DI]	;look-up table
0000:2005	32 E4		XOR AH,AH	;Clear upper byte
0000:2007	B1 04		MOV CL,0004	
0000:2009	D3 C8		ROR AX,CL	;Lower nibble in AH
0000:200B	D2 CC		ROR AH,CL	;Upper nibble in AL
0000:200D	D7		XLAT	;ASCII code of upper
0000:200E	86 E0		XCHG AH,AL	;nibble in AH
0000:2010	D7		XLAT	;ASCII code of lower
0000:2011	CC		INT 03	;nibble in AL
0000:2012	30 31		DB 30,31	;ASCII code look-up
0000:2014	32 33		DB 32,33	;table
0000:2016	34 35		DB 34,35	
0000:2018	36 37		DB 36,37	
0000:201A	38 39		DB 38,39	
0000:201C	41 42		DB 41,42	
0000:201E	43 44		DB 43,44	
0000:2020	45 46		DB 45,46	

Example 4: This is a program to find the factorial of a given number. The program uses the Arithmetic instructions of the 8086/8088 CPU instruction set. The program does not take into account the carry generated by the multiplication and hence the factorial of smaller numbers (up to 8) can be properly calculated.

ADDRESS	OBJECT CODE	LABELS	MNEMONICS	COMMENTS
0000:2000	B8 00 00		MOV AX,0	;Initialize Segment
0000:2003	8E D8		MOV DS,AX	;registers
0000:2005	BE 00 30		MOV SI,3000	
0000:2008	8A 04		MOV AL,[SI]	;Load value from memory
0000:200A	8A 1C	NXT:	MOV BL,[SI]	;Save value and sub.
0000:200C	FE CB		DEC BL	;for repetitive
0000:200E	74 06		JZ OVR	;multiplication
0000:2010	88 1C		MOV [SI],BL	
0000:2012	F7 24		MULW [SI]	
0000:2014	EB F4		JMP NXT	
0000:2016	BF 00 31	OVR:	MOV DI,3100	;Store Computed value
0000:2019	89 05		MOV [DI],AX	;in memory
0000:201B	CC		INT 03	

- ♦ After entering the above program in memory, store the number whose factorial is to be computed at location 0:3000H.
- ♦ Then execute the program from its starting address (0:2000H) and check for the computed value at location 0:3100H. The factorial will be in HEX format
- ♦ The user is urged to modify the program taking into account the carry generated during multiplication of larger numbers and verify the results.

Example 5: This program finds the largest value in a string of data bytes. The string of bytes is pointed by SI register, BH register holds the count value and the result is stored in AL register.

ADDRESS	OBJECT CODE	LABELS	MNEMONICS	COMMENTS
0000:2000	B8 00 00		MOV AX,0000	;Initialize segment
0000:2003	8E D8		MOV DS,AX	;registers
0000:2005	BE 00 21		MOV SI,2100	;Initialize pointer
0000:2008	B7 0A		MOV BH,0A	;Load count value
0000:200A	8A 04		MOV AL,[SI]	;Load byte
0000:200C	46	NXT:	INC SI	
0000:200D	FE CF		DEC BH	;Check 10 bytes in all
0000:200F	74 0A		JE 201B	
0000:2011	8A 1C		MOV BL,[SI]	;Compare current value
0000:2013	3A C3		CMP AL,BL	;with next value
0000:2015	77 F5		JNBE NXT	;If next value is
0000:2017	8A C3		MOV AL,BL	;larger, load it in AL.
0000:2019	EB F1		JMP NXT	;Repeat the process
0000:201B	CC	OVR:	INT 03	;Return to monitor

- ♦ Enter the above program in memory, and store a string of 10 bytes starting from 0:2100H location.
- ♦ Execute the program from its starting address (0:2000H) and check the results. The user may then modify the program taking into account a string of Word values.

10.2 ILLUSTRATION OF ESA 86/88-3 MONITOR ROUTINES

These examples make use of some of ESA 86/88-3 monitor routines listed in Chapter 9. It is recommended that the user go through these routines to get familiar with the usage of these routines.

Example 1: Program to display a message “**HELLO WORLD!**” This program stores the HEX equivalent of the ASCII characters in AL register from memory indexed by SI register.

The program makes use of OUT_CHAR and GET_CHAR routines. If the program is executed in serial mode the message is displayed on the console. In case the program is executed in stand-alone mode, then the output can be observed on the LCD. Then the program waits for the user to enter any key after which it returns to the monitor

ADDRESS	OBJECT CODE	LABELS	MNEMONICS	COMMENTS
0000:2000	BE 00 25		MOV SI,2500	;Set up memory pointer
0000:2003	B1 0D		MOV CL,0E	;and count
0000:2005	8A 04		MOV AL,[SI]	
0000:2007	9A 00 00 00 FE		CALLS 0FE00:0000	;Call OUT_CHAR routine



```

0000:200C  46          INC  SI
0000:200D  E2 F6        LOOP  2005      ;Repeat display
0000:200F  9A A9 00 00 FE CALLS 0FE00:00A9 ;Call GET_CHAR routine
0000:2014  CC          INT   3      ;Return to Monitor

```

The user should fill the locations from 0:2500H to 0:250DH with the following Hexadecimal data bytes. These bytes are the HEX equivalents of the ASCII characters contained in the output message.

```
0000:2500  0A, 0A, 48, 45, 4C, 4C, 4F, 20, 57, 4F, 52, 4C, 44, 21, 20
```

Example 2: This program demonstrates the use of GET_STRING and SEND_STRING routines. The program prompts the user to enter a message from the keyboard in either mode and outputs the same message on the console.

The output of the program will appear only on the console. Execute this program in stand-alone mode, and also connect the trainer to a CRT or PC using RS 232C serial interface. If the communication package is running, the message entered via the PC keyboard interfaced with the trainer will be displayed on the console.

ADDRESS	OBJECT CODE	LABELS	MNEMONICS	COMMENTS
0000:4000	B8 00 00		MOV AX,0000	;Initialize segment
0000:4003	8E C0		MOV ES,AX	;registers
0000:4005	B8 50 40		MOV AX,4050	;Set up memory pointer
0000:4008	9A 31 00 00 FE		CALLS 0FE00:0031	;for message prompt
0000:400D	9A 13 00 00 FE		CALLS 0FE00:0013	;Output message on LCD
0000:4012	9A 31 00 00 FE		CALLS 0FE00:0031	
0000:4017	9A 0E 01 00 FE		CALLS 0FE00:010E	;Call GET_STRING
0000:401C	8B C6		MOV AX,SI	;routine and accept
0000:401E	9A 31 00 00 FE		CALLS 0FE00:0031	;a message
0000:4023	9A AF 01 00 FE		CALLS 0FE00:01AF	;Call SER_SEND_STRING
0000:4028	EA 00 00 00 F0		JMPS 0F000:0000	;routine to output the
				;message onto console
0000:402D			ORG 4050	;Stored message string
0000:4050	45 4E 54 45 52 20		ASC 'ENTER A MESSAGE'	
0000:4056	41 20 4D 45 53 53			
0000:405C	41 47 45			
0000:405F	00		DB 00	

Example 3: This program makes use of some conversion routines like HEX_ASCII and VALID_HEX in addition to other input/output routines. The program waits for the user to enter a valid ASCII character in HEX. If the input is valid (31 – 39 / 41 – 46), the program outputs the equivalent character as a HEX value and repeats the sequence. If the user enters any other value the program displays the message 'INVALID INPUT' and returns control to ESA 86/88-3 monitor. The program may be executed in serial or stand-alone mode of operation.

ADDRESS	OBJECT CODE	LABELS	MNEMONICS	COMMENTS
0000:0000			ORG 2000	
0000:2000	B8 00 00		MOV AX,0000	;Initialise segment
0000:2003	8E C0		MOV ES,AX	;register
0000:2005	9A 31 00 00 FE	RPT:	CALLS 0FE00:0031	;Call OUT_CRLF to
				;go to new line
0000:200A	B8 00 21		MOV AX,2100	;Display message
0000:200D	9A 13 00 00 FE		CALLS 0FE00:0013	;using OUT_STRING



```

0000:2012  9A C7 00 00 FE      CALLS 0FE00:00C7      ;Use GET_BYTE routine
0000:2017  8A D8                MOV  BL,AL           ;to read user input
0000:2019  8A E0                MOV  AH,AL           ;Check if user input
0000:201B  9A 23 01 00 FE      CALLS 0FE00:0123      ;is valid HEX char
0000:2020  3C 00                CMP  AL,00           ;in ASCII using
0000:2022  74 16                JZ   INV             ;VALID_HEX routine
0000:2024  B8 50 21            MOV  AX,2150          ;Display o/p message
0000:2027  9A 13 00 00 FE      CALLS 0FE00:0013      ;if input is valid
0000:202C  8A E3                MOV  AH,BL           ;Convert HEX value
0000:202E  9A 31 01 00 FE      CALLS 0FE00:0131      ;to ASCII using
                                ;ASCII_HEX routine
0000:2033  9A 52 00 00 FE      CALLS 0FE00:0052      ;Output binary value
0000:2038  EB CB                JMP  RPT              ;Repeat the sequence
0000:203A  B8 30 21            INV: MOV  AX,2130      ;For invalid i/p
0000:203D  9A 13 00 00 FE      CALLS 0FE00:0013      ;display appropriate
0000:2042  CC                  INT  3                ;message and return
                                ;control to monitor
0000:2043                        ORG  2100            ;Display message
                                ;strings in memory

0000:2100  45 4E 54 45 52 20  ASC  'ENTER VALID HEX CHARACTER IN ASCII'
0000:2106  56 41 4C 49 44 20
0000:210C  48 45 58 20 43 48
0000:2112  41 52 41 43 54 45
0000:2118  52 20 49 4E 20 41
0000:211E  53 43 49 49
0000:2122  20 00                DB      20,00
0000:2124                        ORG      2130
0000:2130  0A 0D                DB      0A,0D
0000:2132  49 4E 56 41 4C      ASC  'INVALID INPUT'
0000:2137  49 44 20 49 4E
0000:213C  50 55 54 00
0000:2140  00                DB      00H
0000:2141                        ORG      2150
0000:2150  0A 0D                DB      0AH,0D
0000:2152  48 45 58 20 43 48  DB      'HEX CHAR = '
0000:2158  41 52 20 3D 20
0000:215D  20 00                DB      20,00

```

10.3 PROGRAMMING WITH ONBOARD HARDWARE

10.3.1 Use of BREAK key

This program section illustrates the use of BREAK key provided on the trainer. As explained in Chapter 5, this key is connected to NMI or Type 2 interrupt of 8086/8088 CPU. The user has to provide the vectoring information for this interrupt. This program sets the vector locations for the Type 2 interrupt and then waits for the interrupt in a loop. The Code segment of the ISR is set at location 0:000AH as 0000H and the Instruction Pointer is set at location 0:0008H as 2100H.

ADDRESS	OBJECT CODE	LABELS	MNEMONICS	COMMENTS
0000:2000	B8 08 00		MOV AX,0008	;Load ISR offset value
0000:2003	8B F0		MOV SI,AX	;at 0:0008H
0000:2005	B8 00 21		MOV AX,2100	



```

0000:2008  89 04          MOV    [SI],AX
0000:200A  B8 0A 00      MOV    AX,000A    ;Load ISR segment
0000:200D  8B F0        MOV    SI,AX    ;value at 0:000AH
0000:200F  B8 00 00      MOV    AX,0000
0000:2012  89 04          MOV    [SI],AX
0000:2014  8E C0        MOV    ES,AX
0000:2016  B8 50 20      MOV    AX,2050    ;Set up memory
pointer
0000:2019  9A 31 00 00 FE CALLS 0FE00:0031 ;Display message
0000:201E  9A 13 00 00 FE CALLS 0FE00:0013
0000:2023  EB FE          LP: JMP    LP    ;Wait for interrupt

```

After entering the above program, enter the following interrupt service routine at locations 0:2100H onwards. The message string at the end of the routine may be entered using Substitute Memory command also, using the HEX equivalents of the ASCII characters given.

ADDRESS	OBJECT CODE	LABELS	MNEMONICS	COMMENTS
0000:2100	B8 00 00		MOV AX,0000	;Initialize segment
0000:2103	8E C0		MOV ES,AX	;registers
0000:2105	B8 67 20		MOV AX,2067	;Set up memory pointer
0000:2108	9A 31 00 00 FE		CALLS 0FE00:0031	
0000:210D	9A 13 00 00 FE		CALLS 0FE00:0013	;Display message
0000:2112	9A A9 00 00 FE		CALLS 0FE00:00A9	;Wait for user
0000:2117	EA 00 00 00 F0		JMPS 0F000:0000	;strobe and end
0000:211C			ORG 2050	
0000:2050	57 41 49 54 49 4E		ASC 'WAITING FOR'	
0000:2056	47 20 46 4F 52			
0000:205B	0A 0D		DB 0A,0D	
0000:205D	42 52 45 41 4B		ASC 'BREAK KEY'	
0000:2062	20 4B 45 59			
0000:2066	00 0A		DB 00 0A	
0000:2068	42 52 45 41 4B 20		ASC 'BREAK KEY ACCEPTED!'	
0000:206E	4B 45 59 41 43 43			
0000:2074	45 50 54 45 44 21			
0000:207A	00		DB 00	

Now, execute the program from 0:2000H using the GO command. The program will output a message **WAITING FOR BREAK KEY** and will wait for a user strobe. When the user presses the KBINT key on the trainer, the message **BREAK KEY ACCEPTED!** is displayed. The program now waits for another user strobe before returning to command entry mode.

NOTE: The user can independently write vectoring information for Type 2 interrupt at locations 0:0008H and 0:000AH using substitute memory commands. Pressing KBINT key anytime will then transfer control to the program at the address specified by the content of these locations.

10.3.2 Programmable Interrupt Controller 8259A

ESA 86/88-3 provides an onboard Programmable interrupt Controller 8259A that can accept 8 interrupt requests from onboard or off-board sources is used to generate an external interrupt to the CPU on the INTR pin. The configuration for interrupt sources is described in Chapter 2



This program illustrates the use of onboard 8259A to generate interrupts using onboard signals as interrupt sources. The interrupt sources selected here are signals from Port C of 8255-1 at U31. There are two interrupt sources available from this Port on IR1 and IR2 of 8259A. We make use of both the sources in this example.

8259A is programmed with a base value for Interrupt Type 72(decimal). Since the interrupts occur on IR1 and IR2 of the interrupt controller, 8259A will send out a type code of 72+1=73 (decimal) on recognition of IR1. Similarly a type code of 74 (decimal) is sent on recognition of IR2. Thus pointers for interrupt service routines for these interrupts must be stored at location 73x4 (decimal) onwards and at 74x4 (decimal) onwards.

In this example the service routine for IR1 begins at location 0:3050H and the interrupt service routine for IR2 begins at 0:3100H. This vectoring information should be stored at locations 0124H, 0125H, 0126H, and 0127H for IR1 and 0128H, 0129H, 012AH, AND 012BH for IR2. These pointer assignments are set up in the program itself.

The main program now follows:

ADDRESS	OBJECT CODE	LABELS	MNEMONICS	COMMENTS
0000:2500	B8 00 00		MOV AX,0000	;Initialize Segment
0000:2503	8E C0		MOV ES,AX	;registers
0000:2505	26		ES	
0000:2506	C7 06 24 01 50 30		MOV @0124,3050	;Set up vectoring
0000:250C	26		ES	;information for
0000:250D	C7 06 26 01 00 00		MOV @0126,0000	;interrupt
0000:2513	26		ES	;service routine
0000:2514	C7 06 28 01 00 31		MOV @0128,3100	
0000:251A	26		ES	
0000:251B	C7 06 2A 01 00 00		MOV @012A,0000	
0000:2521	BA F4 FF		MOV DX,0FFF4	;Initialize 8259 PIC
0000:2524	B0 13		MOV AL,13	
0000:2526	EE		OUT DX,AL	
0000:2527	BA F6 FF		MOV DX,0FFF6	
0000:252A	B0 48		MOV AL,48	;Base value type: 72d
0000:252C	EE		OUT DX,AL	
0000:252D	B0 03		MOV AL,03	;Automatic EOI 8086/88
0000:252F	EE		OUT DX,AL	;Mode
0000:2530	B0 F9		MOV AL,0F9	;Mask all interrupts
0000:2532	EE		OUT DX,AL	;except INT 1 and 2
0000:2533	B0 80		MOV AL,80	
0000:2535	BA E6 FF		MOV DX,0FFE6	;Initialize 8255-1 for
0000:2538	EE		OUT DX,AL	;generating interrupt
0000:2539	BA E4 FF		MOV DX,0FFE4	
0000:253C	B0 0F	BAC:	MOV AL,0F	;signals
0000:253E	EE		OUT DX,AL	;Output high & low
0000:253F	B0 00		MOV AL,00	;signals since 8259 is
0000:2541	EE		OUT DX,AL	;initialized for
				;edge triggered
				;interrupt recognition
0000:2542	FB		STI	
0000:2543	EB F0		JMP BAC	;Repeat continuously

The above routine continuously generates High bits on Port C of 8255-1, which can be used as interrupt sources. In order to generate the interrupts, the following jumper settings are necessary.

JP 16: B – C for IR1

JP 17: B – C for IR2

The interrupt service routines are given below. The program displays one of the following messages depending on which jumper is shorted in the above manner.

INT 1 RECOGNISED (if JP 16 is shorted) or

INT 2 RECOGNISED (if JP 17 is shorted)

ADDRESS	OBJECT CODE	LABELS	MNEMONICS	COMMENTS
0000:3050			ORG 3050	
0000:3050	FA		CLI	;Clear interrupt flag
0000:3051	B8 00 30		MOV AX,3000	;Display message for
0000:3054	9A 13 00 00 FE		CALLS 0FE00:0013	;interrupt 1
0000:3059	CF		IRET	
0000:3100			ORG 3100	
0000:3100	FA		CLI	;Clear interrupt flag
0000:3101	B8 16 30		MOV AX,3012	;Display message for
0000:3104	9A 13 00 00 FE		CALLS 0FE00:0013	;interrupt 2
0000:3109	CF		IRET	
0000:3000			ORG 3000	
0000:3000			DB 0D	
0000:3001			ASC 'INT 1 RECOGNISED '	
0000:3011			DB 00	
0000:3012			DB 0D	
0000:3013			ASC 'INT 2 RECOGNISED '	
0000:3014			DB 00	

10.3.3 Programmable Interval Timer, 8253

ESA 86/88-3 provides the user with three independent timing channels via an onboard Programmable Interval Timer 8253A. All the timers are fully available to the user. The addressing information for controlling and sending data to this programmable interval timer is given in Chapter 6. Clock, Gate and Output signals of these timers are brought out on a 10-pin header J10.

The following program module initializes TIMER 1 of 8253A in Mode 3 (Square Wave Generator). After entering and executing this program the user can observe a square waveform at TIMER1 of connector J10 on an oscilloscope with a time base of 5ms. Note that the user should connect appropriate signals to CLK1 and GATE1 terminals of connector J10.

ADDRESS	OBJECT CODE	LABELS	MNEMONICS	COMMENTS
0000:2000	BA FF FF		MOV DX,0FFFF	
0000:2003	B0 76		MOV AL,76	;Control Word for
0000:2005	EE		OUT DX,AL	;TIMER 1 in MODE 3.
0000:2006	BA FB FF		MOV DX,0FFFB	;Load 16-bit Count
0000:2009	B0 10		MOV AL,10	;in TIMER 1 Count
0000:200B	EE		OUT DX,AL	;register
0000:200C	EE		OUT DX,AL	



0000:200D EB FE

NOW: JMP NOW

10.3.4 Onboard Programmable Peripheral Interface, 8255

ESA 86/88-3 provides the user with 48 programmable I/O lines using two programmable peripheral interfaces, viz. 8255 ICs at U31 and U42. These lines are brought to 26-pin connectors J8 and J9 respectively. The user may connect any interface module compatible to these connectors and program the corresponding PPI to work with the interface. The addresses of control and data ports are given in Chapter 6.

This is a demonstration program for Stepper Motor Interface assumed to be connected over connector J8 of the trainer (corresponding to 8255-PPI Low.) The interface can be obtained from ESA Pvt. Ltd., Bangalore as an option.

ADDRESS	OBJECT CODE	LABELS	MNEMONICS	COMMENTS
0000:2000	BA E6 FF		MOV DX,0FFE6	;Initialize all 8255
0000:2003	B0 80		MOV AL,80	;Ports as output
0000:2005	EE		OUT DX,AL	
0000:2006	BA E0 FF		MOV DX,0FFE0	
0000:2009	B0 88		MOV AL,88	;Output data to ports
0000:200B	EE		OUT DX,AL	
0000:200C	E8 04 00		CALL 2013	;Introduce delay
0000:200F	D0 C8		ROR AL,1	;Rotate data byte for
0000:2011	EB F8		JMP 200B	;rotation of motor
0000:2013	B9 00 40		MOV CX,4000	;Delay subroutine
0000:2016	E2 FE	RPT: LOOP	RPT	
0000:2018	C3		RET	

Enter the program from 0:2000H onwards, execute it from this location and observe the behavior of the stepper motor. Change the data byte at 0:2010 from C8H to C0H and observe the results.

10.4 USE OF 8087 CO-PROCESSOR

ESA 86/88-3 provides direct support for an optional Numeric Data processor-8087. To utilize this feature, the user has to simply install 8087 IC in the socket provided (U32). No other hardware changes are required. The following two examples illustrate the use of 8087.

NOTE: ESA 86/88-3 Symbolic One-line Assembler does not support NDP instructions. So these programs must be entered directly in machine code (hexadecimal values) using Substitute Memory Command.

Example 1: The following program assumes that two 32-bit integer data values, a and b, are stored at locations 3000H and 3004H onwards respectively. It then computes a result C = square root of (a^2+b^2) and stores the result as a 32-bit integer starting at location 3008H. (Code segment value is assumed as 0000)

ADDRESS	OBJECT CODE	LABELS	MNEMONICS	COMMENTS
0000:2000	BB 00 30		MOV BX,3000	;Point to 'a'
0000:2003	DB 07		FLD (BX)	;Load 'a'



0000:2005	9B	FWAIT	
0000:2006	DA 0F	FMUL (BX)	;Compute a2
0000:2008	9B	FWAIT	
0000:2009	DD D9	FST ST (1)	;Save a2 in ST (1)
0000:200B	9B	FWAIT	
0000:200C	81 D3 04 00	ADD BX,0004	;Point to 'b'
0000:2010	DB 07	FLD (BX)	;Load 'b'
0000:2012	9B	FWAIT	
0000:2013	DA 0F	FMUL (BX)	;Compute b2
0000:2015	9B	FWAIT	
0000:2016	D8 C1	FADD ST(1)	;ST(0) = a ² +b ²
0000:2018	9B	FWAIT	
0000:2019	D9 FA	FSQRT	;ST(0) = Sqrt (a ² +b ²)
0000:201B	9B	FWAIT	
0000:201C	81 D3 04 00	ADD BX,0004	;Point to location
0000:2020	DB 17	FST (BX)	;for C and store the
0000:2022	9B	FWAIT	;result as 32-bit
0000:2023	CC	INT 3	;integer in C & exit

1. Load the above program into memory.
2. Set up data values a and b as follows.

```
0000:3000: 03, 00, 00, 00
0000:3004: 04, 00, 00, 00
```

3. Execute the program and observe the result. It should be as follows:

```
0000:3008: 05, 00, 00, 00
```

4. The user is urged to try with different data values to observe the results.

Example 2: The following program calculates Sin (Z) where Z is in degrees and $0 < Z < 90$. Note that the values 0 and 90 are not allowed. They must be handled separately.

Sin (Z) is calculated using the Tangent function (FPTAN) of 8087 as follows:
Suppose $\tan (Z/2) = Y/X$. Then $\sin (Z) = 2XY/(X^2 + Y^2)$

The FPTAN function of 8087 requires the argument, in radians on the top of the stack and returns the result as Y/X where X is the top of stack and Y is the next element. Further the argument must satisfy the condition $0 < \text{argument} < \pi/4$. Hence Z is restricted to satisfy $0 < Z < 90$. This restriction can be eliminated at the cost of more computation.

If Z be unrestricted, then the FPREM function of 8087 can be used to reduce the argument to the required range and use the relation $\sin (\pi+Z) = -\sin (Z)$

In the following program, the argument Z is first divided by 2 (preparatory to using the relation described above), then converted into radians and then the tangent is calculated. The resulting values X and Y are used to calculate sin (Z).

The input to the program, Z (in degrees) must be set up in the register AL, as a Hex value. The output, in packed BCD form is available in the register AX. A decimal point is to be assumed before the first BCD digit.



ADDRESS	OBJECT CODE	LABELS	MNEMONICS	COMMENTS
0000:2000	B4 00		MOV AH,00	
0000:2002	D1 E8		SHR AX,1	;Z=Angle = Angle/2
0000:2004	A3 5A 20		MOV Arg1,AX	;Preparatory for
0000:2007	9B DF 06 5A 20		FLDI Arg1	;loading into NDP.
0000:200C	C7 06 5C 20 B4 00		MOV Arg2,00B4	;(Arg2) = 180
0000:2012	9B DF 06 5C 20		FLDI Arg2	;Insert a FWAIT and
				;load Arg2.
0000:2017	9B DE F9		FDIVRP	;Wait and divide (ST)
				;= Z/180
0000:201A	9B D9 EB		FLDPI	;(ST)=PI
0000:201D	9B DE C9		MULRP	;(ST)=PI*Z/80 = angle
				;in radians
0000:2020	9B D9 F2		FPTAN	;Compute tan as Y/X
0000:2023	9B D9 C0		FLD ST(0)	;Copy X onto stack top
0000:2026	9B D8 C9		FMUL ST,ST(1)	;ST = X*X
0000:2029	9B D9 C2		FLD ST(2)	;ST = Y
0000:202C	9B D8 CB		FMUL ST,ST(3)	;ST = Y*Y
0000:202F	9B DE C1		FADDP ST(1),ST	;Y*Y is popped off &
				;ST = X*X+ Y*Y
0000:2032	9B DE F9		FDIVRP	;ST = X/(X*X+Y*Y)
0000:2035	9B DE C9		FMULRP	;ST = X*Y/CX*X+Y*Y)
0000:2038	9B D9 C0		FLD ST(0)	;Copy value onto stack
0000:203B	9B DE C1		FADDP ST(1),ST	;ST = 2*X*Y[X*X-Y*Y)
0000:203E	C7 06 5E 20 10 27		MOV Arg3,2710H	
0000:2044	9B DF 06 5E 20		FLDI Arg3	
0000:2049	9B DE C9		FMURP	;Result = 10000*sin(Z)
0000:204C	9B D9 FC		FRNDINT	;Round to integer
0000:204F	9B DF 36 60 20		FSTP Arg 4	;and store as BCD
0000:2054	9B		FWAIT	;Get the result into
0000:2055	A1 60 20		MOV AX,Arg 4	;Accumulator
0000:2058	CC		INT 3	;Return to monitor

Note that 16 locations 0:205A to 206A (ARG1, ARG2, ARG3 and ARG4) are used for storing the constants and results.

1. Load the above program into the memory.
2. Set up the input parameter in the register AL (for example, to calculate Sin 60, (AL) = 3C H)
3. Execute the program and observe the contents of AX (with input as (AL) = 3C, (AX) will now be 8660 so Sin 60 = 0.8660)
4. Repeat the program with different input data and observe the output as shown below.

Input(AL)	Output(AX)	Calculated function
1E	5000	sin 30
2E	7193	sin 46
10	2756	sin 16
56	9976	sin 86

The user may try using the FPREM function of 8087, calculate sin Z for any value of Z.



10.5 PROGRAMMING WITH INTERFACES

This section consists of programming examples for onboard interfaces that may be used with ESA 86/88-3. The user can use these programming modules to test the interfaces as well as to develop individual projects.

10.5.1 Onboard 8-Bit DAC

This is a demonstration program for ESA 86/88-3 onboard DAC to generate triangular waveform. The user may observe the waveform at Test Point J13 on trainer after entering and executing the following program. Please refer to Chapter 6 for a detailed discussion on onboard DAC.

ADDRESS	OBJECT CODE	LABELS	MNEMONICS	COMMENTS
			DLY EQU 6030	;Initialize delay
				;routine address
			ORG 6000	
0000:6000	BA D6 FF		MOV DX,0FFD6	;Configure 8255 Ports
0000:6003	B0 80		MOV AL,80	;as output Ports
0000:6005	EE		OUT DX,AL	
0000:6006	B9 FF 00	AGN:	MOV CX,0FF	;Count for +ve slope
0000:6009	B8 00 00		MOV AX,0000	
0000:600C	BA D0 FF	UP:	MOV DX,0FFD0	;Output digital value
0000:600F	EE		OUT DX,AL	;from Port A
0000:6010	FE C0		INC AL	
0000:6012	E8 1B 00		CALL DLY	;Introduce small delay
0000:6015	E2 F8		LOOP UP	
0000:6017	B9 FF 00		MOV CX,0FF	;Count for -ve slope
0000:601A	BA D0 FF	DN:	MOV DX,0FFD0	;Output digital value
0000:601D	EE		OUT DX,AL	;from Port A
0000:601E	FE C8		DEC AL	
0000:6020	E2 F8		CALL DLY	;Introduce small delay
0000:6023	E9 E4 FF		LOOD DN	
0000:6025	EB ED		JMP AGN	;Repeat forever.
0000:6027			ORG DLY	
0000:6030	90		NOP	
0000:6031	90		NOP	
0000:6032	90		NOP	
0000:6033	C3		RET	

10.5.2 Onboard 12-Bit ADC

This is a demonstration program for the optional onboard Eight Channel 12bit A/D Converter for both multi-channel & single channel operation.

When the ADC is operated in single channel mode, the MUX at U25 need not be populated and jumpers JP18 & JP19 are open. Analog input is applied at screw terminal TP. When the ADC is to be used for multi-channel operation, the MUX is populated and eight channels are available as selected by the channel select lines. Jumper JP19 is closed in this case and the analog signals are applied at the screw terminals designated as CH0 to CH7 provided at J12.



Note: When the ADC is in multi-channel mode no signal is applied at TP. Refer Chapter 6 for a detailed discussion on ADC operating modes (Bipolar, Uni-polar), Voltage ranges, etc.

ADDRESS	OBJECT CODE	MNEMONICS	COMMENTS
0000:2500	B8 00 00	MOV AX,0000	;Initialise Segment
0000:2503	8E D8	MOV DS,AX	;registers
0000:2505	8E C0	MOV ES,AX	
0000:2507	BE 00 20	MOV SI,2000	
0000:250A	B0 00	MOV AL,00	
0000:250C	88 04	MOV [SI],AL	
0000:250E	BA D7 FF	MOV DX,FFD7	;Initialise 8255-3
0000:2511	B0 80	MOV AL,80	;ports as o/p ports
0000:2513	EE	OUT DX,AL	;and 8255-4 Port A as
0000:2514	BA D6 FF	MOV DX,FFD6	;O/p, Port B as I/p
0000:2517	B0 83	MOV AL,83	;Port C upper as O/p
0000:2519	EE	OUT DX,AL	;Port C lower as I/p
0000:251A	BA D1 FF	MOV DX,FFD1	;Disable latches
0000:251D	B0 03	MOV AL,03	
0000:251F	EE	OUT DX,AL	
0000:2520	8A 04	MOV AL,[SI]	
0000:2522	B9 04 00	MOV CX,0004	
0000:2525	D2 C0	ROL AL,CL	;Output Channel value
0000:2527	0C 80	OR AL,80	;to MUX
0000:2529	BA D4 FF	MOV DX,FFD4	
0000:252C	EE	OUT DX,AL	
0000:252D	B9 20 00	MOV CX,0020	;MUX settling time
0000:2530	E2 FE	LOOP 2530	;delay
0000:2532	BA D1 FF	MOV DX,FFD1	;Start conversion
0000:2535	B0 02	MOV AL,02	;pulse to ADC
0000:2537	EE	OUT DX,AL	
0000:2538	B0 03	MOV AL,03	
0000:253A	EE	OUT DX,AL	
0000:253B	B0 00	MOV AL,00	
0000:253D	EE	OUT DX,AL	
0000:253E	90	XCHG AX,AX	;ADC conversion time
0000:253F	90	XCHG AX,AX	;delay
0000:2540	90	XCHG AX,AX	
0000:2541	90	XCHG AX,AX	
0000:2542	B0 01	MOV AL,01	;Enable latches for
0000:2544	EE	OUT DX,AL	;capturing data
0000:2545	2B C0	SUB AX,AX	
0000:2547	BA D4 FF	MOV DX,FFD4	
0000:254A	EC	IN AL,DX	;Read digital value
0000:254B	24 0F	AND AL,0F	;from latches and
0000:254D	8A E0	MOV AH,AL	;store in AX register
0000:254F	BA D2 FF	MOV DX,FFD2	
0000:2552	EC	IN AL,DX	
0000:2553	8B D8	MOV BX,AX	
0000:2555	EB 2B	JMP 2582	
0000:2557	8A 0C	MOV CL,[SI]	;Read next channel
0000:2559	9A A9 00 00 FE	CALLS FE00:00A9	;using GET_CHAR
0000:255E	3C 2C	CMP AL,2C	;If char = '\,'
0000:2560	74 06	JE 2568	;increment Ch. No.
0000:2562	3C 2D	CMP AL,2D	;If char = '\-'



```

0000:2564 74 0F          JE      2575          ;decrement Ch. No.
0000:2566 EB EF          JMP      2557          ;Wait for valid i/p
0000:2568 FE C1          INC      CL              ;Routine to update
0000:256A 80 F9 08       CMP      CL,08          ;ADC channel
0000:256D 75 02          JNE      2571
0000:256F B1 00          MOV      CL,00
0000:2571 88 0C          MOV      [SI],CL
0000:2573 EB A5          JMP      251A          ;Repeat conversion
0000:2575 80 F9 00       CMP      CL,00
0000:2578 75 04          JNE      257E
0000:257A B1 07          MOV      CL,07
0000:257C EB F3          JMP      2571
0000:257E FE C9          DEC      CL
0000:2580 EB EF          JMP      2571
0000:2582 9A 31 00 00 FE CALLS    FE00:0031      ;Routines for
0000:2587 B8 00 21       MOV      AX,2100      :displaying Channel
0000:258A 9A 13 00 00 FE CALLS    FE00:0013      ;Nos and digital
0000:258F 8A 04          MOV      AL,[SI]      ;outputs
0000:2591 9A 52 00 00 FE CALLS    FE00:0052
0000:2596 B8 0F 21       MOV      AX,210F
0000:2599 9A 13 00 00 FE CALLS    FE00:0013
0000:259E 8B C3          MOV      AX,BX
0000:25A0 9A 6A 00 00 FE CALLS    FE00:006A      ;Display WORD value
0000:25A5 EB B0          JMP      2557          ;Repeat sequence

```

Enter the above program from 0:2500H onwards using ESA 86/88-3 Symbolic one-Line Assembler. The following look up table may now be entered using either the DB and ASC directives at location 0:2100H onwards.

```

0000:2100          DB 0A,0D
0000:2102          ASC 'CHANNEL NO: '
0000:210E          DB 00
0000:210F          DB 0A,0D
0000:2111          ASC 'DIGITAL VALUE: '
0000:211E          DB 00

```

- ♦ Execute the program from 0:2500H onwards in either mode of operation.
- ♦ Give the analog input signal at the desired channel and select the particular channel by sifting through the channels.
- ♦ Enter ',' (comma) to increment the current channel value and '-' (minus) to decrement the current channel value. If multi-channel mode is selected, the program outputs the digital equivalent of the analog voltage at the channel. If single channel operation is selected, then the program outputs the digital equivalent of the analog voltage at TP.
- ♦ Observe the results after changing the configurations for the different operating modes available with the ADC. (Ref Chapter 6)

10.5.3 Parallel Printer Interface

This program demonstrates direct output to the printer using PRINT_STRING routine wherein a parallel printer interfaced with the trainer can be accessed regardless of the printer enable DIP switch SW6. The printer cable is connected over the 25-pin D-type female connector J7. The necessary cable may be obtained from ESA Pvt Ltd, Bangalore as an option.



This program sends a message "HELLO WORLD!" directly to the printer. This routine can be called from the user's program when the system is operating in either of the two modes. Refer Chapter 8 for a description of this routine and its calling address.

ADDRESS	OBJECT CODE	LABELS	MNEMONICS	COMMENTS
0000:2500	B8 00 00		MOV AX,0000	;Initialize segment
0000:2503	8E D8		MOV DS,AX	;registers
0000:2505	BE 00 30		MOV SI,3000	;Set up memory
				;pointer to message
0000:2508	9A F3 01 00 FE		CALLS 0FE00:01F3	;Call PRINT_STRING
0000:250D	CC		INT 3	;routine and
				;terminate

Along with the above program, enter the following look up table for the message string from locations 0:3000H. Then, execute the program and observe the output string printed by the printer.

```
0000:3000 0A,0D,20,20,20,48,45,4C,4C,4F,20,57,4F,52,4C,44,21,00
```

This program makes use of the monitor routine to print the line of characters. Alternatively the user may write his/her program to output to the printer by initializing the 8255 at U1 and making use of the information given in Section 6.10

10.5.4 Liquid Crystal Display Interface

This example illustrates the initialization of the optional onboard LCD available with ESA 86/88-3 Trainer. The LCD is controlled by the 8255 positioned at U27. The details of the instruction and data word formats for LCD may be obtained from Crystalonics Displays User's Manual.

This program displays a string "HELLO WORLD!" on the LCD continuously with a specific delay. The user can utilize this program as a model to develop his own projects using the LCD. This program can be executed in either mode of trainer operation.

ADDRESS	OBJECT CODE	LABELS	MNEMONICS	COMMENTS
0000:2000			CMD EQU 2034	
0000:2000			STR EQU 2026	
0000:2000			DWR EQU 204A	
0000:2000			MES EQU 2008	
0000:2000	B0 80		MOV AL,80	;Initialise all 8255
0000:2002	BA DE FF		MOV DX,0FFDE	;ports as output
0000:2005	EE		OUT DX,AL	
0000:2006	EB 0D		JMP 2015	
0000:2008	48 45 4C 4C 4F 20	MES:	ASC 'HELLO WORLD!'	
0000:200E	57 4F 52 4C 44 21			;Message string
0000:2014	00		DB 00	
0000:2015	B0 0C	START:	MOV AL,0C	;Display ON, No
0000:2017	E8 1A 00		CALL CMD	;Cursor or Blinking
				;character
0000:201A	B0 01		MOV AL,01	;Clear Display
0000:201C	E8 15 00		CALL CMD	
0000:201F	2E		CS	
0000:2020	8D 16 08 20		LEA DX,@MES	;Set up pointer to



```

0000:2024  8B DA          MOV    BX,DX      ;message string
0000:2026  8A 07          STR: MOV    AL,[BX]
0000:2028  3C 00          CMP    AL,00      ;Check for string
0000:202A  74 E9          JE     START      ;termination
                                ;character
0000:202C  E8 1B 00        CALL   DWR          ;Call routine to
0000:202F  43             INC    BX          ;write data
0000:2030  EB F4          JMP    STR
0000:2032  EB E1          JMP    START        ;Repeat continuously
0000:2034  BA D8 FF        CMD: MOV    DX,0FFD8 ;Write into Data
0000:2037  EE             OUT    DX,AL        ;register
0000:2038  BA DC FF        MOV    DX,0FFDC    ;Enable Read/Write
0000:203B  B0 06          MOV    AL,06        ;in Instruction
0000:203D  EE             OUT    DX,AL        ;register
0000:203E  B0 08          MOV    AL,08
0000:2040  EE             OUT    DX,AL
0000:2041  B0 00          MOV    AL,00
0000:2043  EE             OUT    DX,AL
0000:2044  B9 00 80        MOV    CX,8000
0000:2047  E2 FE          DY1: LOOP   DY1
0000:2049  C3             RET
0000:204A  BA D8 FF        DWR: MOV    DX,0FFD8 ;Routine to write
0000:204D  EE             OUT    DX,AL        ;data
0000:204E  B0 F4          MOV    AL,0F4
0000:2050  BA DC FF        MOV    DX,0FFDC
0000:2053  EE             OUT    DX,AL
0000:2054  B0 F2          MOV    AL,0F2        ;Return to start of
0000:2056  EE             OUT    DX,AL        ;LCD display
0000:2057  B0 FA          MOV    AL,0FA
0000:2059  EE             OUT    DX,AL
0000:205A  B0 F2          MOV    AL,0F2
0000:205C  EE             OUT    DX,AL
0000:205D  B9 00 80        MOV    CX,8000      ;Delay between
0000:2060  E2 FE          DY2: LOOP   DY2      ;repeated displays
0000:2062  C3             RET

```

Enter the program from 0:2000H location onwards. Execute the program and observe the output on the LCD

This chapter covered a variety of programming examples that were developed using ESA 86/88-3 Symbolic One-line assembler. The user is urged to try different variations of the program modules given here, so that the capabilities of the programming environment provided are fully appreciated. We welcome any recommendations or suggestions for improvement.

CONNECTOR DETAILS

The connector details for interfacing peripherals and signal definitions of these connectors are described in this section. A brief summary of the connectors available on the trainer is described below. Refer the component layout diagram in Appendix C to locate these connectors.

J1 & J2	:	50 Pin Bus connectors connected to system bus
J3	:	4 –terminal power connector
J4	:	9 pin, D-type, female connector for RS 232 C compatible serial communication
J5	:	9 pin, D-type, male connector for RS 485 compatible serial communication.
J6	:	PC Keyboard Connector.
J7	:	25 Pin Parallel Printer interface Connector
J8 & J9	:	26 pin parallel I/O Connector
J10	:	10 pin Header for 8253 timer signals
J11	:	15 pin Flow-strip for LCD Module
J12	:	9- Screw terminals for ADC channel selection
J13	:	DAC output Test Point

J1 : CONNECTED TO SYSTEM BUS

PIN NO	SIGNAL	PIN NO	SIGNAL
1	GND	2	GND
3	+5V	4	+5V
5	+5V	6	+5V
7	GND	8	GND
9	GATE1	10	GATE2
11	CONCLK	12	RESET
13	N.C	14	PCLK
15	CINT0*	16	CINT1*
17	CINT2*	18	CINT3*
19	CINT4*	20	CINT5*
21	CINT6*	22	CINT7*
23	LS0	24	INTA*
25	LS1	26	MWTC*
27	LS2	28	IOWC*
29	NC	30	CAS0
31	NC	32	CAS1
33	LOCK*	34	CAS2
35	MRDC*	36	AMWTC*
37	IORC*	38	AIOWC*
39	ALE	40	EXTRDY
41	CLK1	42	CLK2
43	GND	44	GND
45	+5V	46	+5V
47	TIMER1	48	TIMER2
49	GND	50	GND



J2: CONNECTED TO SYSTEM BUS

PIN NO	SIGNAL	PIN NO	SIGNAL
1	GND	2	GND
3	BA0	4	CBHE *
5	BA1	6	BA2
7	BA3	8	BA4
9	BA5	10	BA6
11	BA7	12	BA8
13	GND	14	BA9
15	BA10	16	BA11
17	BA12	18	BA13
19	BA14	20	BA15
21	BA16	22	GND
23	BA17	24	BA18
25	BA19	26	NC
27	NC	28	CRTC
29	NC	30	GND
31	BD0	32	BD1
33	BD2	34	BD3
35	BD4	36	BD5
37	BD6	38	BD7
39	GND	40	GND
41	BD8	42	BD9
43	BD10	44	BD11
45	BD12	46	BD13
47	BD14	48	BD15
49	GND	50	GND

J3: POWER CONNECTOR

COLOUR CODE	SUPPLY TERMINAL
BLACK	GROUND
RED	+5 V
YELLOW	+12V
GREEN	-12V

J4 & J5: SERIAL INTERFACE CONNECTORS (Refer Appendix E)

J6: PC KEYBOARD DIN CONNECTOR

PIN NO.	SIGNAL
1	KBDCLK (through 100Ω resistance)
2	GND
3	KBD DATA(through 100Ω resistance)
4	VCC
5	Not Connected

J7: 25-PIN PARALLEL PRINTER INTERFACE CONNECTOR

PIN NO	SIGNAL	PIN NO	SIGNAL
1	P3C4	2	P3B0
3	P3B1	4	P3B2
5	P3B3	6	P3B4
7	P3B5	8	P3B6
9	P3B7	10	NC
11	P3C3	12	NC
13	NC	14	NC
15	NC	16	NC
17	NC	18	GND
19	GND	20	GND
21	GND	22	GND
23	GND	24	GND
25	GND		

J8 & J9: 26 PIN PARALLEL I/O CONNECTORS**J8 CONNECTED TO 8255 at U31 (EVEN)**

PIN No	SIGNAL	PIN No	SIGNAL
1	P1C4	2	P1C5
3	P1C2	4	P1C3
5	P1C0	6	P1C1
7	P1B6	8	P1B7
9	P1B4	10	P1B5
11	P1B2	12	P1B3
13	P1B0	14	P1B1
15	P1A6	16	P1A7
17	P1A4	18	P1A5
19	P1A2	20	P1A3
21	P1A0	22	P1A1
23	P1C6	24	P1C7
25	+5V	26	GND



J9 CONNECTED TO 8255 at U42 (ODD)

PIN NO	SIGNAL	PIN NO	SIGNAL
1	P2C4	2	P2C5
3	P2C2	4	P2C3
5	P2C0	6	P2C1
7	P2B6	8	P2B7
9	P2B4	10	P2B5
11	P2B2	12	P2B3
13	P2B0	14	P2B1
15	P2A6	16	P2A7
17	P2A4	18	P2A5
19	P2A2	20	P2A3
21	P2A0	22	P2A1
23	P2C6	24	P2C7
25	+5V	26	GND

J10: HEADER FOR 8253 TIMER SIGNALS

PIN NO	SIGNAL	PIN NO	SIGNAL
1	VCC	2	CLK 0
3	TIMER 0	4	GATE 0
5	CLK 2	6	TIMER 2
7	GATE 2	8	CLK 1
9	GATE 1	10	TIMER 1

J11: FLOW-STRIP FOR 20 x 4 LCD MODULE

PIN NO	SIGNAL	PIN NO	SIGNAL
1	GND	2	VCC
3	GND	4	P5C1
5	P5C2	6	P5C3
7	P5A0	8	P5A1
9	P5A2	10	P5A3
11	P5A4	12	P5A5
13	P5A6	14	P5A6
15	BK_LT		



ASCII CODES

HEX	DEC	CHAR	HEX	DEC	CHAR
00	0	NUL	22	34	“
01	1	SOH	23	35	#
02	2	STX	24	36	\$
03	3	ETX	25	37	%
04	4	EOT	26	38	&
05	5	ENQ	27	39	!
06	6	ACK	28	40	(
07	7	BEL	29	41	)
08	8	BS	2A	42	*
09	9	HT	2B	43	+
0A	10	LF	2C	44	‘
0B	11	VT	2D	45	-
0C	12	FF	2E	46	.
0D	13	CR	2F	47	/
0E	14	SO	30	48	0
0F	15	SI	31	49	1
10	16	DLE	32	50	2
11	17	DC1	33	51	3
12	18	DC2	34	52	4
13	19	DC3	35	53	5
14	20	DC4	36	54	6
15	21	NAK	37	55	7
16	22	SYN	38	56	8
17	23	ETB	39	57	9
18	24	CAN	3A	58	:
19	25	EM	3B	59	;
1A	26	SUB	3C	60	<
1B	27	ESC	3D	61	=
1C	28	FS	3E	62	>
1D	29	GS	3F	63	?
1E	30	RS	40	64	@
1F	31	US	41	65	A
20	32	SP	42	66	B
21	33	!	43	67	C



HEX	DEC	CHAR	HEX	DEC	CHAR
44	68	D	62	98	b
45	69	E	63	99	c
46	70	F	64	100	d
47	71	G	65	101	e
48	72	H	66	102	f
49	73	I	67	103	g
4A	74	J	68	104	h
4B	75	K	69	105	I
4C	76	L	6A	106	j
4D	77	M	6B	107	k
4E	78	N	6C	108	l
4F	79	O	6D	109	m
50	80	P	6E	110	n
51	81	Q	6F	111	o
52	82	R	70	112	p
53	83	S	71	113	q
54	84	T	72	114	r
55	85	U	73	115	s
56	86	V	74	116	t
57	87	W	75	117	u
58	88	X	76	118	v
59	89	Y	77	119	w
5A	90	Z	78	120	x
5B	91	[	79	121	y
5C	92	\	7A	122	z
5D	93	]	7B	123	{
5E	94	^	7C	124	l
5F	95	-	7D	125	}
60	96	‘	7E	126	~
61	97	a	7F	127	DEL

SERIAL COMMUNICATION CABLE REQUIREMENTS

1. RS 232C Cable requirements

ESA 86/88-3 requires a null modem cable in order to communicate with other systems. RS 232C Standard communication can be established by connecting an interfacing cable to the Host computer system or CRT terminal through J4, the 9-pin D-type female connector provided on the trainer.

The connections required for the RS 232C Cable are as follows.

Trainer End (9-pin D-type female)	System End
TXD(Pin 3)	RXD (Pin 3)
RXD(Pin 2)	TXD (Pin 2)
RTS(Pin 7)	CTS (Pin 5)
CTS(Pin 8)	RTS (Pin 4)
DSR(Pin 6)	DTR (Pin 20)
DTR(Pin 4)	DSR (Pin 6)
GND(Pin 5)	GND (Pin 7)

Note: If hardware handshaking is not required interconnect the following

- RTS and CTS (Pin 7 and 8)
- DSR and DTR (Pin 6 and 4)

2. RS 485 Communication Cable requirements

RS 485 Standard communication can be established by connecting an interfacing cable to the Host computer system or CRT terminal through J5, the 9-pin D-type male connector provided on the trainer. These are the connections required for the RS 485 cable

Trainer End (9-pin D-type male)	System End
A (Pin 3 & 2)	RXD ⁺ & TXD ⁺
B (Pin 8 & 6)	RXD ⁻ & TXD ⁻



ESA 86/88-3 SYMBOLIC ONE-LINE ASSEMBLER **MNEMONICS SYNTAX**

This appendix gives the mnemonic syntax of each 8086/8088 CPU instruction supported by ESA 86/88-3 Symbolic One-Line Assembler. Comments about certain instructions are also given at appropriate locations. Refer Chapter 8 for a detailed discussion of the facilities provided by the assembler.

The following convention is used for indicating registers, addresses and immediate data

Reg8 : 8-bit registers AL, CL, DL, BL, AH, CH, BH, DH

Reg16 : 16-Bit registers AX, CX, DX, BX, SP, BP, SI, DI

SegReg : Segment Registers DS, ES, SS, CS

Mem8* : Effective address or Symbolic reference pointing to Byte data

Mem16* : Effective address or Symbolic reference pointing to Word data

Data8 : 8-bit Immediate data or Symbolic reference

Data16 : 16-bit Immediate data or Symbolic reference

Disp16 : 16-bit displacement (Offset)

Seg16 : 16-bit Segment specifier

Port8 : I/O Port with 8-bit address

Address : Memory location (Offset) at which to branch program control.

Short label: Label representing 8-bit displacement i.e. within ± 127 bytes from current IP.

Label : Label representing displacement beyond ± 127 bytes from current IP, but in the same segment.

Far label : Label referencing an address in another Code Segment

*Effective addresses pointing to either Byte or Word values can be included in the source statements in any of the following formats.

1. Base Register.
2. Index Register
3. Base + Index
4. Base + Displacement
5. Index + Displacement
6. Base + Index + Displacement
7. @Address or @Label

Instruction Mnemonic	Description	Operands	Syntax / Remarks
AAA	ASCII Adjust after	-	AAA



	Addition		
AAD	ASCII Adjust after Division	-	AAD
AAM	ASCII Adjust after Multiplication	-	AAM
AAS	ASCII Adjust after Subtraction	-	AAS
ADC	ADD Destination and Source with carry	Reg8,Reg8 Reg8,Mem8 Reg8,Data8 Mem8,Reg8	Byte Mode: ADCB
		Mem8,Data8	ADCB
		Reg16,Reg16 Reg16,Mem16 Reg16,Data16 Mem16,Reg16	Word Mode: ADCW
		Mem16,Data16	ADCW
ADD	ADD Destination and Source	Reg8,Reg8 Reg8,Mem8 Reg8,Data8 Mem8,Reg8	Byte Mode: ADDB
		Mem8,Data8	ADDB
		Reg16,Reg16 Reg16,Mem16 Reg16,Data16 Mem16,Reg16	Word Mode: ADDW
		Mem16,Data16	ADDW
AND	Logical AND operation	Reg8,Reg8 Reg8,Mem8 Reg8,Data8 Mem8,Reg8	Byte Mode: ANDB
		Mem8,Data8	ANDB

		Reg16,Reg16 Reg16,Mem16 Reg16,Data16 Mem16,Reg16	Word Mode: ANDW
		Mem16,Data16	ANDW
CALL NEAR	CALL an intra-segment procedure/ subroutine	Disp16 / Mem16 / Label Reg16 @Disp16 / @Label	CALL
CALL FAR	CALL inter-segment procedure/ subroutine	Seg16:Disp16 Mem16 Label @Disp16	CALLS
CBW	Convert Byte to Word	-	CBW
CLC	Clear Carry Flag	-	CLC
CLD	Clear Direction Flag	-	CLD
CLI	Clear Interrupt enable Flag	-	CLI
CMC	Complement Carry Flag	-	CMC
CMP	Compare Source with Destination value	Reg8,Reg8 Reg8,Mem8 Reg8,Data8 Mem8,Reg8	Byte Mode: CMPB
		Mem8,Data8	CMPB
		Reg16,Reg16 Reg16,Mem16 Reg16,Data16 Mem16,Reg16	Word Mode: CMPW
		Mem16,Data16	CMPW
CMPS	Compare Source and Destination Strings	Destination String , Source String	Byte Mode CMPSB
			Word Mode CMPSW
CS	Code Segment override prefix	-	CS
CWD	Convert Word to Double	-	CWD



	Word		
DAA	Decimal Adjust after Addition	-	DAA
DAS	Decimal Adjust after Subtraction	-	DAS
DEC	Decrement by one	Reg8	DECB
		Mem8	DECB
		Reg16	DECW
		Mem16	DECW
DIV	Divide Unsigned Word by Byte or Double Word by Word	Reg8	DIVB
		Mem8	DIVB
		Reg16	DIVW
		Mem16	DIVW
DS	Data Segment override prefix	-	DS
ES	Extra Segment override prefix	-	ES
ESC	Escape to external co-processor	Not applicable	External co-processor mnemonics are not currently supported. The mnemonic is only displayed during disassembly.
HLT	Halt until interrupt or Reset	-	HLT
IDIV	Divide Signed Word or Double Word	Reg8	IDIVB
		Mem8	IDIVB
		Reg16	IDIVW
		Mem16	IDIVW
IMUL	Multiply Signed Byte or Word	Reg8	IMULB
		Mem8	IMULB
		Reg16	IMULW
		Mem16	IMULW
IN	Input Byte or Word from port	AL,DX AL,Port8	INB

		AX,DX AX,Port8	INW
INC	Increment by one	Reg8	INCB
		Mem8	INCB
		Reg16	INCW
		Mem16	INCW
INT	Interrupt program execution	Interrupt Type	INT
INTO	Interrupt program on overflow	-	INTO
IRET	Return from interrupt service routine	-	IRET
JA/JNBE	Jump if Above / Not below or equal	Address / Short label	JA/JNBE
JAE/JNB	Jump if above or equal / Not below	Address / Short label	JAE/JNB
JB/JNAE	Jump if below / Not above or equal	Address / Short label	JB/JNAE
JBE/JNA	Jump if below or equal / Not above	Address / Short label	JBE/JNA
JC	Jump on Carry	Address / Short label	JC
JCXZ	Jump if CX = 0	Address / Short label	JCXZ
JE/JZ	Jump if equal / Jump on Zero	Address / Short label	JE/JZ
JG/JNLE	Jump if greater / Not lesser or equal	Address / Short label	JG/JNLE
JGE/JNL	Jump if greater or equal / Not lesser	Address / Short label	JGE/JNL
JL/JNGE	Jump if lesser / Not greater or equal	Address / Short label	JL/JNGE
JLE/JNG	Jump if lesser or equal / Not greater	Address / Short label	JLE/JNG
JMP NEAR	Jump Intra-segment / Indirect	Address / Label / Disp16 @Address / @Label Reg16 Mem16	JMP
JMP FAR	Jump Inter-segment	Address (Seg16:Disp16) Far Label	JMPS

		@Address Reg16 Mem16	
JNC	Jump if no Carry	Address / Short label	JNC
JNE / JNZ	Jump if not equal / on non-zero	Address / Short label	JNE / JNZ
JNO	Jump if no overflow	Address / Short label	JNO
JNP / JPO	Jump if no parity / Odd parity	Address / Short label	JNP / JPO
JNS	Jump if not sign (Sign flag = 0)	Address / Short label	JNS
JO	Jump on overflow	Address / Short label	JO
JP / JPE	Jump on Parity / Even Parity	Address / Short label	JP / JPE
JS	Jump if Sign flag = 1	Address / Short label	JS
LAHF	Load AH with flag register	-	LAHF
LDS	Load DS and register from memory	Reg16,Mem16	LDS
LEA	Load Effective address	Reg16,Reg16 Reg16,Mem16	LEA (The instruction is assembled without any pre-determined Segment override prefix. Segment override, if any must be explicitly entered and assembled.)
LES	Load ES and register from memory	Reg16,Mem16	LES
LOCK	Lock bus	-	LOCK
LODS	Load string into accumulator	-	Byte Mode LODSB
			Word Mode LODSW
LOOP	Loop till CX = 0	Address / Short label	LOOP
LOOPE / LOOPZ	Loop while Zero flag = 1 and CX $\neq$ 0	Address / Short label	LOOPE / LOOPZ
LOOPNE / LOOPNZ	Loop while Zero flag = 0 and CX $\neq$ 0	Address / Short label	LOOPNE / LOOPNZ



MOV	Copy Byte or Word from source register / memory to destination register / memory location.	Reg8,Reg8 Reg8,Data8 Reg8,Mem8 Mem8,Reg8	Byte Mode MOVB
		Mem8,Data8	MOVB
		Reg16,Reg16 Reg16,Data16 Reg16,Mem16 Mem16,Reg16 SegReg,Reg16 SegReg,Mem16 Reg16,SegReg Mem16,SegReg	Word Mode MOVW
		Mem16,Data16	MOVW
MOVS	Copy String from source to destination	-	Byte Mode MOVSB
			Word Mode MOVSW
MUL	Multiply unsigned Byte or Word	Reg8	MULB
		Mem8	MULB
		Reg16	MULW
		Mem16	MULW
NEG	Negate Byte or Word and add 1 (2's complement)	Reg8	NEGB
		Mem8	NEGB
		Reg16	NEGW
		Mem16	NEGW
NOP	No operation (only fetch and decode)	-	NOP (Encoded as XCHGW AX,AX)
NOT	Logical NOT operation	Reg8	NOTB
		Mem8	NOTB
		Reg16	NOTW
		Mem16	NOTW

OR	Logical OR operation	Reg8,Reg8 Reg8,Mem8 Reg8,Data8 Mem8,Reg8	Byte Mode: ORB
		Mem8,Data8	ORB
		Reg16,Reg16 Reg16,Mem16 Reg16,Data16 Mem16,Reg16	Word Mode: ORW
		Mem16,Data16	ORW
OUT	Output Byte or Word to port	DX,AL Port8,AL	OUTB
		DX,AX Port8,AX	OUTW
POP	Pop from top of stack	SegReg / Reg16 / Mem16	POP (POP CS is not allowed)
POPF	Pop from top of stack to flags register	-	POPF
PUSH	Push to top of stack	SegReg / Reg16 / Mem16	PUSH
PUSHF	Push flag registers to top of stack	-	PUSHF
RCL	Rotate left through carry flag	Reg8,CL Reg8,1	RCLB
		Mem8,CL Mem8,1	RCLB
		Reg16,CL Reg16,1	RCLW
		Mem16,CL Mem16,1	RCLW
RCR	Rotate right through carry flag	Reg8,CL Reg8,1	RCRB



		Mem8,CL Mem8,1	RCRB
		Reg16,CL Reg16,1	RCRW
		Mem16,CL Mem16,1	RCRW
REP/REPE/REPZ	Repeat following instruction until CX = 0 or Zero flag $\neq$ 1	-	REP/REPE/REPZ
REPNE/REPNZ	Repeat following instruction until CX = 0 or Zero flag = 1	-	REPNE/REPNZ
RET	Return from procedure	No Operand / Data16	RET
RET FAR	Return from far procedure (inter-segment)	No Operand / Data16	RETS
ROL	Rotate Left	Reg8,CL Reg8,1	ROLB
		Mem8,CL Mem8,1	ROLB
		Reg16,CL Reg16,1	ROLW
		Mem16,CL Mem16,1	ROLW
ROR	Rotate Right	Reg8,CL Reg8,1	RORB
		Mem8,CL Mem8,1	RORB
		Reg16,CL Reg16,1	RORW
		Mem16,CL Mem16,1	RORW
SAHF	Store AH in flag register byte	-	SAHF

SAL/SHL	Arithmetic / Logical left shift	Reg8,CL Reg8,1	SALB SHLB
		Mem8,CL Mem8,1	SALB SHLB
		Reg16,CL Reg16,1	SALW SHLW
		Mem16,CL Mem16,1	SALW SHLW
SAR	Arithmetic Right Shift	Reg8,CL Reg8,1	SARB
		Mem8,CL Mem8,1	SARB
		Reg16,CL Reg16,1	SARW
		Mem16,CL Mem16,1	SARW
SBB	Subtract Byte or Word with Borrow	Reg8,Reg8 Reg8,Mem8 Reg8,Data8 Mem8,Reg8	Byte Mode: SBBB
		Mem8,Data8	SBBB
		Reg16,Reg16 Reg16,Mem16 Reg16,Data16 Mem16,Reg16	Word Mode: SBBW
		Mem16,Data16	SBBW
SCAS	Scan String	-	Byte Mode SCASB
			Word Mode SCASW
SHR	Logical Right Shift	Reg8,CL Reg8,1	SHRB



		Mem8,CL Mem8,1	SHRB
		Reg16,CL Reg16,1	SHRW
		Mem16,CL Mem16,1	SHRW
SS	Stack Segment override prefix	-	SS
STC	Set Carry flag	-	STC
STD	Set Direction flag	-	STD
STI	Set Interrupt Enable flag	-	STI
STOS	Store into String	-	Byte Mode STOSB
			Word Mode STOSW
SUB	Subtract	Reg8,Reg8 Reg8,Mem8 Reg8,Data8 Mem8,Reg8	Byte Mode: SUBB
		Mem8,Data8	SUBB
		Reg16,Reg16 Reg16,Mem16 Reg16,Data16 Mem16,Reg16	Word Mode: SUBW
		Mem16,Data16	SUBW
TEST	Logical compare by AND operation on bits and updating flag bits without changing the operand.	Reg8,Reg8 Reg8,Mem8 Reg8,Data8 Mem8,Reg8	Byte Mode: TESTB
		Mem8,Data8	TESTB
		Reg16,Reg16 Reg16,Mem16 Reg16,Data16 Mem16,Reg16	Word Mode: TESTW

		Mem16,Data16	TESTW
WAIT	Wait until TEST pin is asserted	-	WAIT
XCHG	Exchange Words or Bytes	Reg8,Reg8 Mem8,Reg8	Byte Mode: XCHGB
		Reg16,Reg16 Mem16,Reg16	Word Mode: XCHGW
XLAT	Translate AL value using look-up table in memory	-	XLAT
XOR	Logical Exclusive OR	Reg8,Reg8 Reg8,Mem8 Reg8,Data8 Mem8,Reg8	Byte Mode: XORB
		Mem8,Data8	XORB
		Reg16,Reg16 Reg16,Mem16 Reg16,Data16 Mem16,Reg16	Word Mode: XORW
		Mem16,Data16	XORW

